For Want of a Nail: Parser Bugs in the Pwnies

Sergey Bratus

For Want of a Nail...

"... for want of a nail, the shoe was lost;
for want of a shoe the horse was lost;
and for want of a horse the rider was lost."

For Want of a Nail...

"... for want of a nail, the shoe was lost;
for want of a shoe the horse was lost;
and for want of a horse the rider was lost."

For Want of a Nail...

"... for want of a nail, the shoe was lost;
for want of a shoe the horse was lost;
and for want of a horse the rider was lost."

a parser was lost...

...a pwnie was gained!

Bugs that got famous, 2013-2014

- Simple, **avoidable** parser bugs
- Simple, **avoidable** protocol implementation bugs

Bugs that got famous, 2012-2014

- Simple, **avoidable** parser bugs
- Simple, **avoidable** protocol implementation bugs

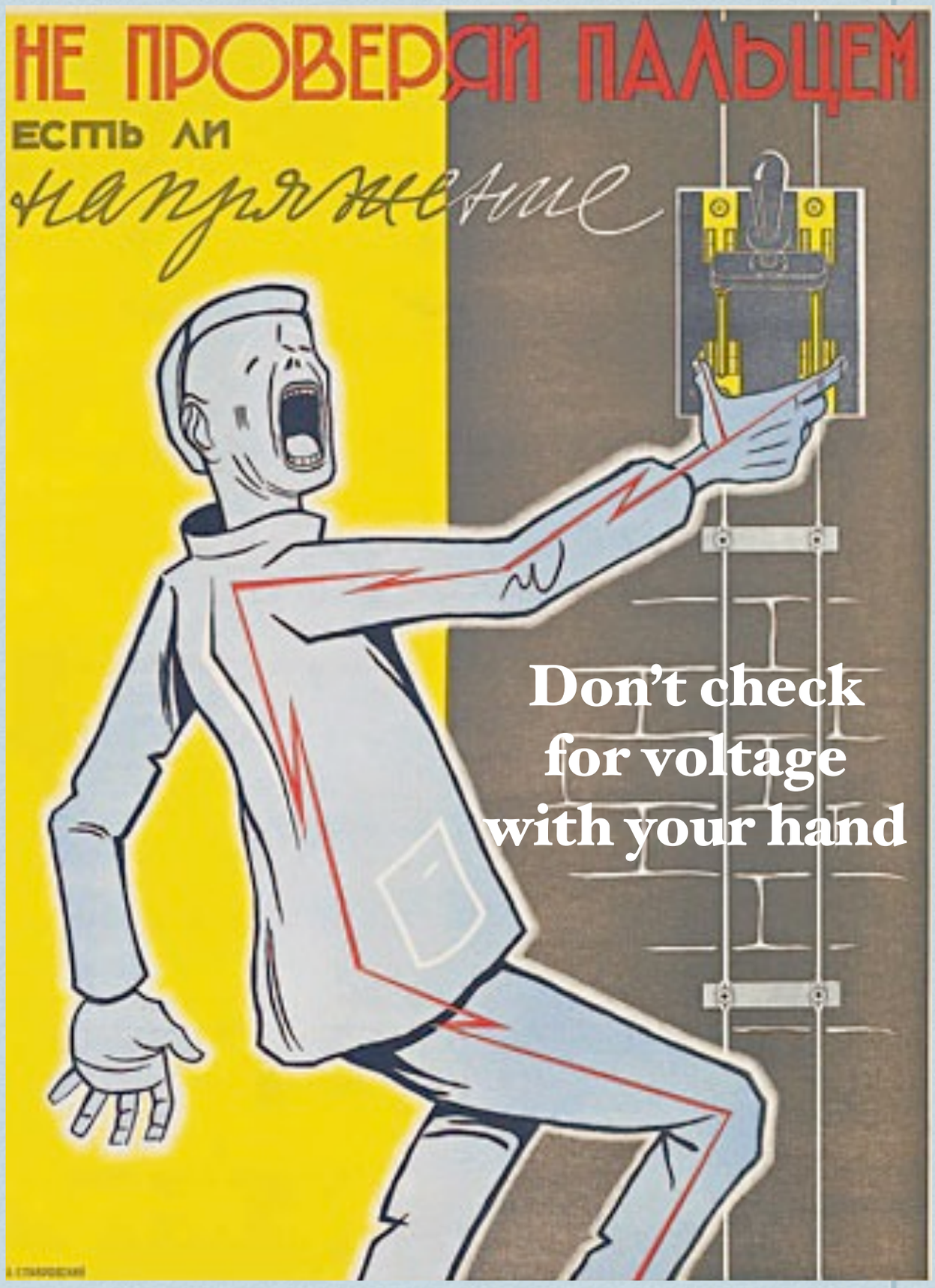
Bugs that got famous, 2012-2014

- What went wrong?
- Why it went wrong?
- What tools & practices would have **prevented** it from going wrong?
- How to choose such tools & practices?
- How to eliminate these **classes** of bugs?
  - What is the right **model** for input parsing, validation, processing?

Hindsight is 20/20, right?

So are workplace safety rules!

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




КИРПИЧ
УКЛАДЫВАЙ
ПРАВИЛЬНО
25
Эфф


КИРПИЧ
УКЛАДЫВАЙ
ПРАВИЛЬНО
25
1906


НАВЕРХУ
РАБОТАЮТ
НЕ
СТОЙ
ПОД МАЧТОЙ

КИРПИЧ

Й

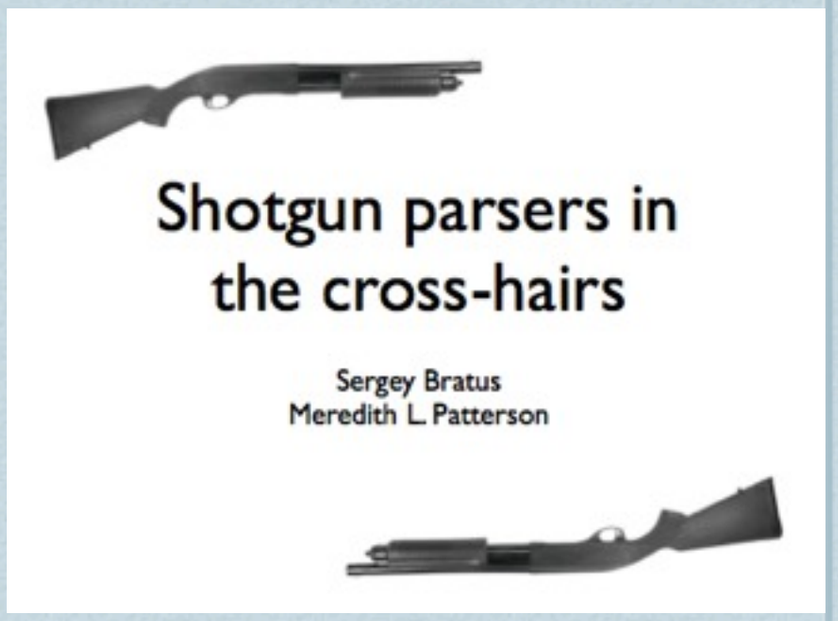
НЕ ВЕДИ РАБОТУ ПОДКОЛОМ

НАВЕРХУ РАБОТАЮТ

НЕ СТОЙ ПОД МАЧТОЙ

An input-handling anti-pattern

- "***Shotgun parsers***": an input-handling anti-pattern
  - **validity** of protocol field values checked in **ad-hoc** order
  - checks **interspersed** with **malloc**, **memcopy**, arithmetic
  - worst when syntax is **complex** & **context-sensitive**



BIND 8.4	1999
OpenSSH 3.3	2002
OpenBSD 4.0	2007

BruCON 2012, video at langsec.org/

Oldies but goodies

- BIND 8.4 DNS NXT record overflow, **1999**, by ADM
  - buffer for hostname overflown; to find hostname length, parser must chase back across RRs
- OpenSSH 3.3 pre-auth challenge/response, **2002**, by Gobbles
  - variable option lengths across a packet must sum up to length of the packet; integer overflowed before packet ended
- OpenBSD 4.0 ICMPv6, **2007**, by Core Security
  - IPv6 fragment chains vs *BSD mbuf heap chains

Recognition vs Processing

Checks

Input validation

Recognition

malloc( )

memcpy( )

+, - , *, /

Recognition vs Processing

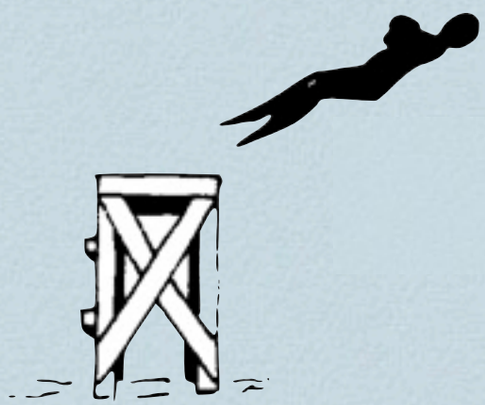
"Sanity Checks"

+, - , *, /

malloc( )

"Input sanitization"

memcpy( )



Common patterns: context-sensitive syntax

- **Context-sensitive** syntax features
  - **redundant** fields, **dependent** values
    - **nested**, variable-length fields
    - values must agree **across** layers/objects
- Simple condition **assumed** checked
- Memory allocation, read or write happens before syntax is fully checked => a **weird machine** emerges

Context-sensitive syntax + ad-hoc parsers = pwnage

So what's new?

2013-2014

So what's new?

2013-2014

So what's new?

2013-2014

goto fail;

So what's new?

2013-2014

goto fail;

So what's new?

2013-2014

goto fail;

chunked

- Heartbleed, OpenSSL, 2014
- Android Master Key, 2013
- goto fail; Apple 2014
- Nginx chunked encoding, CVE-2013-2028

- compare with Apache CVE-2002-3092

goto fail;

NGINX™

chunked

Heartbleed is a parser bug!

Heartbeat sent to victim

SSLv3 record:

Length

`SSL3_RECORD`

`HeartbeatMessage`

Type	Length	Payload data
TLS1_HB_REQUEST		1 byte

Heartbleed is a parser bug!

Heartbeat sent to victim

SSLv3 record:

Length

SSL3_RECORD

HeartbeatMessage

Type	Length	Payload data
TLS1_HB_REQUEST		1 byte

Heartbleed is a parser bug!

Heartbeat sent to victim

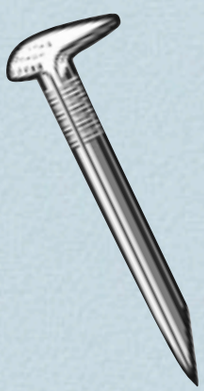
SSLv3 record:

SSL3_RECORD

Length
4 bytes

HeartbeatMessage

Type	Length	Payload data
TLS1_HB_REQUEST	65535 bytes	1 byte



Heartbleed is a parser bug!

Heartbeat sent to victim

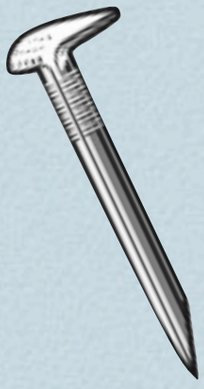
SSLv3 record:

Length
4 bytes

`SSL3_RECORD`

`HeartbeatMessage`

Type	Length	Payload data
TLS1_HB_REQUEST	65535 bytes	1 byte



Victim's response

SSLv3 record:

Length
65538 bytes

HeartbeatMessage:

Type	Length	Payload data	
TLS1_HB_RESPONSE	65535 bytes	65535 bytes	

Heartbleed is a parser bug!

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

Heartbeat sent to victim

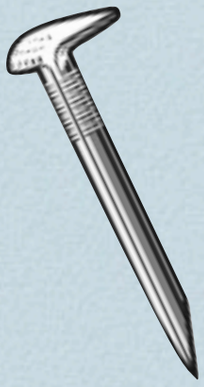
SSLv3 record:

Length
4 bytes

HeartbeatMessage:

Type	Length	Payload data	
TLS1_HB_REQUEST	65535 bytes	1 byte	

Must agree, but never checked



Victim's response

SSLv3 record:

Length
65538 bytes

```
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
```

HeartbeatMessage:

Type	Length	Payload data	
TLS1_HB_RESPONSE	65535 bytes	65535 bytes	

```
-       /* 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!
ОСТОРОЖНЕЕ С ЛОПАТОЙ

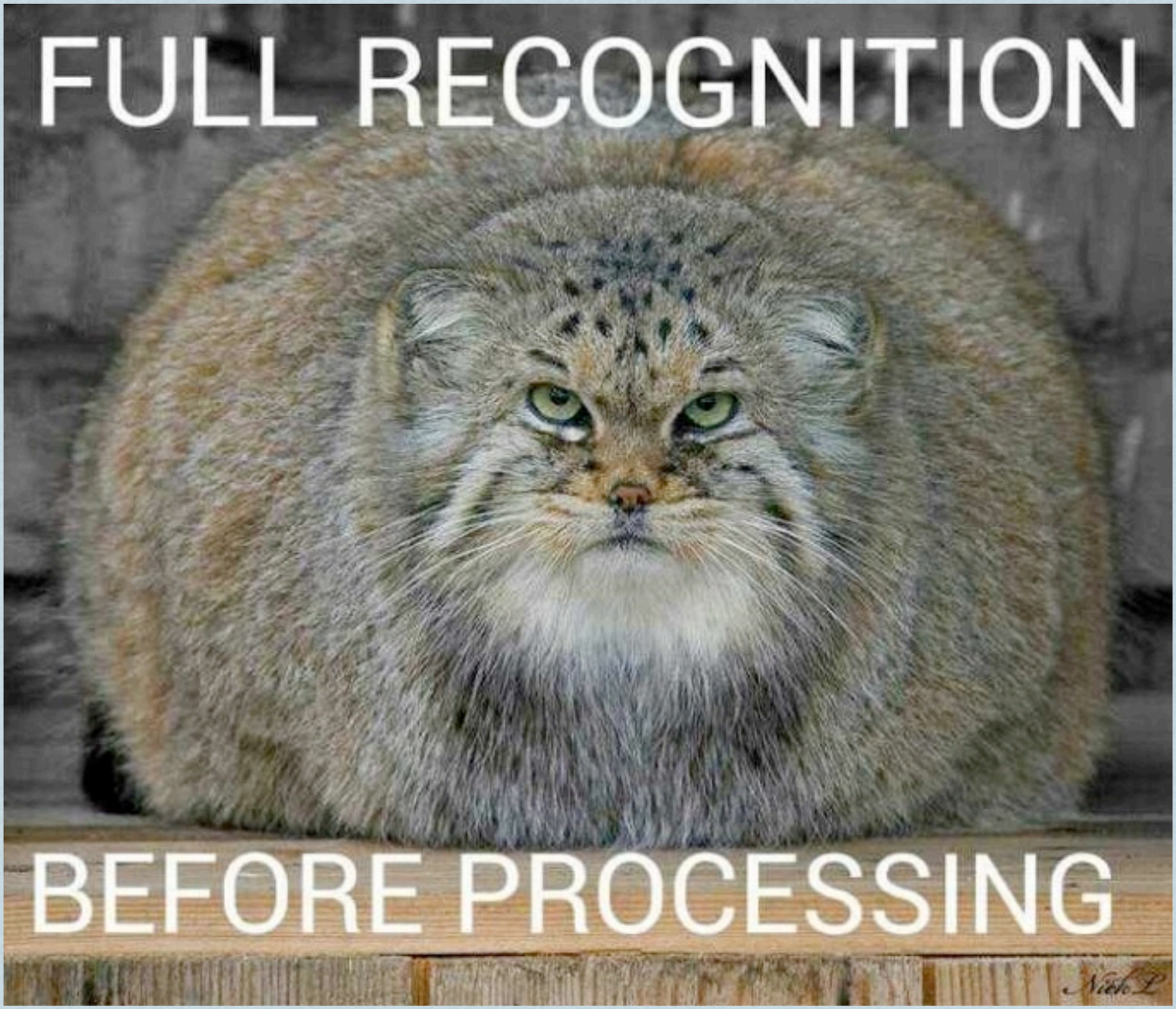
Your input is a language;
treat it as such:
write a grammar spec.

Parser code should read like
the grammar.

Nested length fields are context-sensitive 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

LangSec.org cat says:



```
/*

MANUL THE LANGSEC CAT SAYS:

       _ _
 ◄/       \►                 ___ _   _ _    _
 ⋗== ò ▾ ó ==⋖               | __| | | | |  | |
   \   △  /    __            | _|| |_| | |__| |__
  /       \  |  \            |_|  \___/|____|____|
 /         \ |   |                                  before processing
|  |        | ||  |   ___ ___ ___ ___   ___ _  _ ___ ____ ___ ___  _  _
 \ |  ||  |  /  /    | _ \ __/ __/ _ \ / __| \| |_ _|_  /| __/ _ \| \| |
  \_oo__oo_/___/     |   / _| (_| (_) | (_ | .` || | | | | _| (_) | .` |
                     |_|_\___\___\___/ \___|_|\_|___| |_| |___\___/|_|\_|


*/
```

utf-8 manul by

Melissa Μέλισσα
@0xabad1dea
#126,030,998

goto fail;

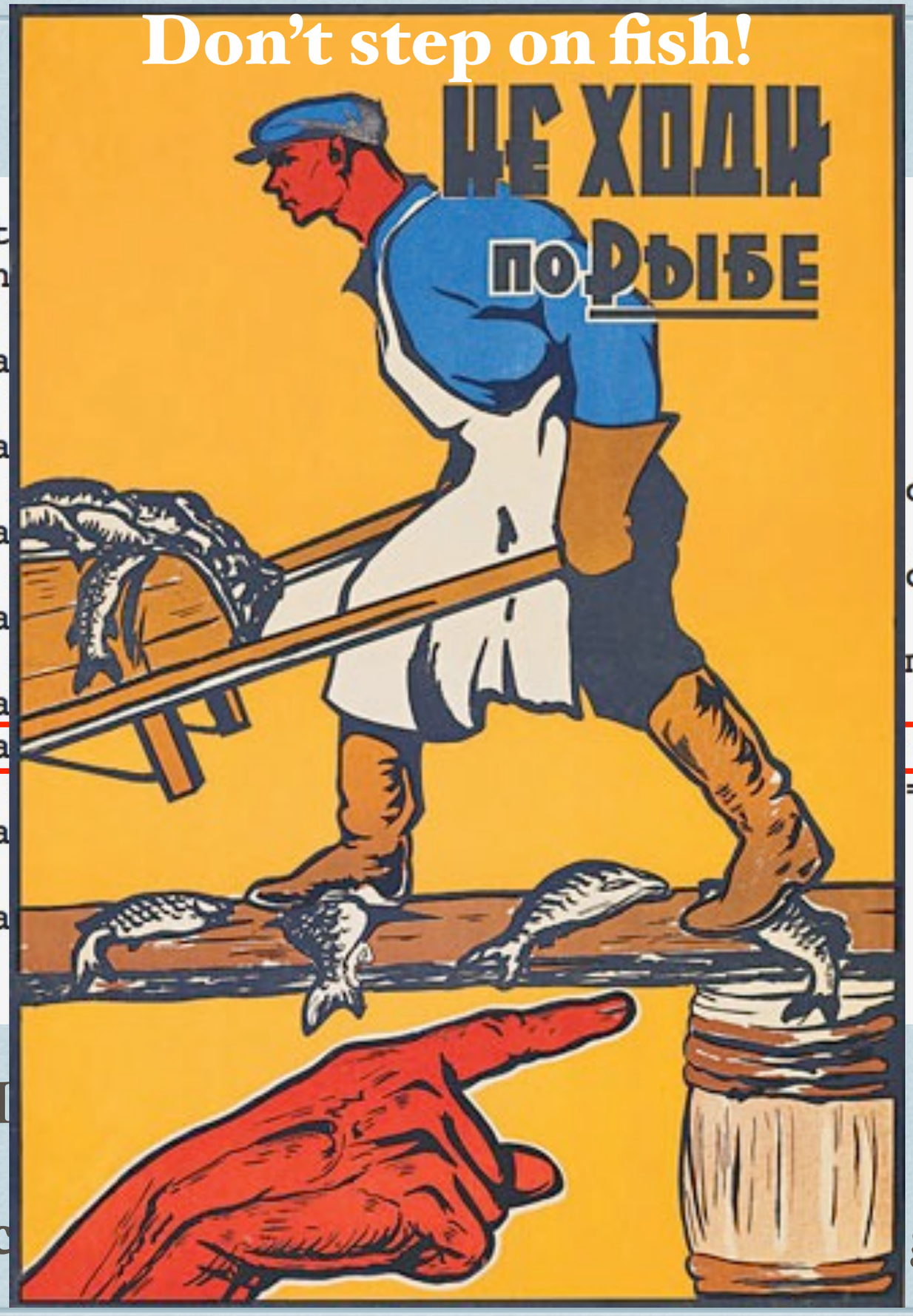
```
. . .
hashOut.data = hashes + SSL_MD5_DIGEST_LEN;
hashOut.length = SSL_SHA1_DIGEST_LEN;
if ((err = SSLFreeBuffer(&hashCtx)) != 0)
    goto fail;
if ((err = ReadyHash(&SSLHashSHA1, &hashCtx)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &clientRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
    goto fail;
    goto fail;  /* MISTAKE! THIS LINE SHOULD NOT BE HERE */
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
    goto fail;

err = sslRawVerify(...);
. . .
```

- Apple's SSL state machine, **hand-coded**
- **State machine done wrong: code must be generated!**

```
. . .
hashOut.dat
hashOut.len
if ((err =
    goto fa
if ((err =
    goto fa
if ((err =                                          om)) != 0)
    goto fa
if ((err =                                          om)) != 0)
    goto fa
if ((err =                                          ms)) != 0)
    goto fa
    goto fa                                          HERE */
if ((err =                                          = 0)
    goto fa

err = sslRa
. . .
```



- Apple’s SSI
- **State mac** **generated!**

An aside: GnuTLS Hello

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

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

https://github.com/azet/CVE-2014-3466_PoC/blob/master/poc.py

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       = '''
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
'''                             # Random (gmt_unix_time + random bytes)
HS_SessID_Len   = 'c8'          # Session ID Length 200 Bytes (!)
HS_SessID_Data  = '''
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
'''                             # Session ID Data (Payload)
```

```
# 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       = '''
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
'''                             # Random (gmt_unix_time + random bytes)
HS_SessID_Len   = 'c8'          # Session ID Length 200 Bytes (!)
HS_SessID_Data  = '''
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
'''                             # 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('\n', '').decode('hex')
```

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

# Handshake Protocol: ServerHello
HS_Type         = '02'          # Handshake
HS_Length       = '00 00 f6'    # 246 Bytes
HS_Version      = '03 01'       # TLS 1.0
HS_Random       = '''
53 8b 7f 63 c1 0e 1d 72 0a b3 f8 a7 0f f5 5d
65 58 42 80 c1 fb 4f db 9a aa 04 a3 d3 4b 71
'''                             # Random (gm
HS_SessID_Len   = 'c8'          # Session ID
HS_SessID_Data  = '''
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff ff
'''                                 # Session ID

MaliciousServerHello = (
    R_Type      + R_Version     + R_Length
    HS_Type     + HS_Length     + HS_Versio
    HS_Random   + HS_SessID_Len + HS_SessID_
).replace(' ', '').replace('\n', '').decode(
```

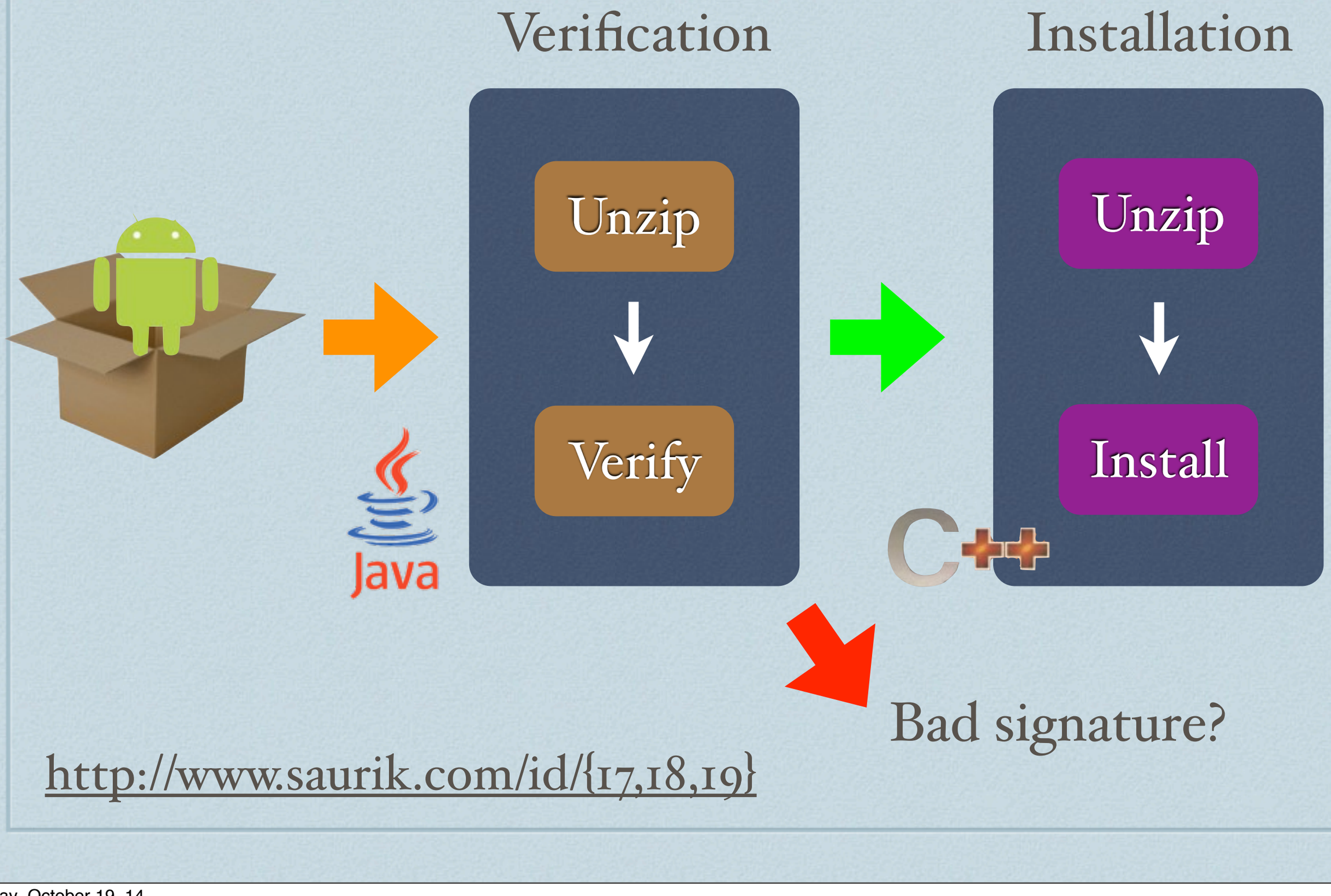
Parser differentials

- Two parsers, one message ...

  two different parses!

- We've seen this before in:
  - "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection", Ptacek & Newsham, 1998
  - X.509 certs: "PKI layer cake", Kaminsky, Sassaman, Patterson, 2010

Android Master Key: Parser Differentials



http://www.saurik.com/id/{17,18,19}

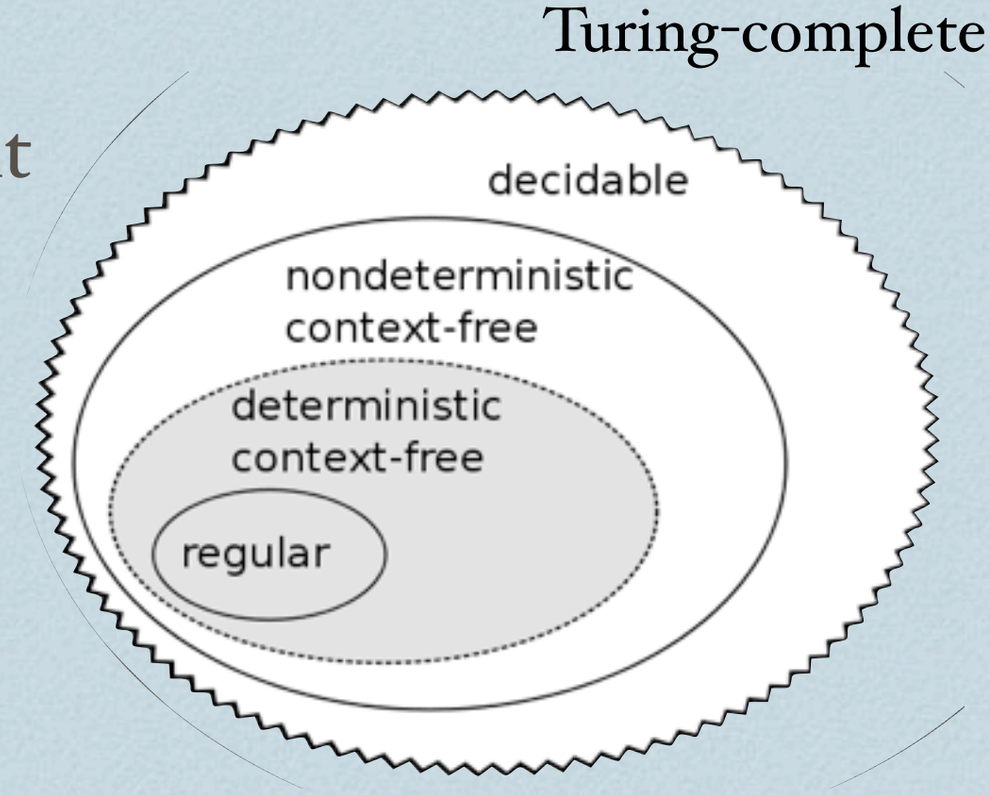
Android Master Key: A Parser Differential

- 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}

Android Master Key: A Parser Differential

- Initial fixes still kept two different parsers, just patched them.
  - Parser equivalence is **UNDECIDABLE** beyond deterministic context free languages



- Finally fixed right: the **same** parser used for both verification & installation, not two **different** parsers

Androi ntial

lation

zip

tall

http://ww re?

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
```

Apache CVE-2002-3092

```
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";
```

~~19~~ **DEADBEEF**

```
A bunch of data broken up
D
into chunks.
0
```

Apache CVE-2002-3092

```
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 ";
```

~~19~~ **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) {
```

Ap 92

```
foreach my
    my $re
    $reque
    $reque
    $reque
    $reque
    $reque
    $reque
```

```
--- http_pro
+++ http_pro
@@ -2171,7 +
     /* Othe
-    len_to_
+    len_to_
remaining;
     len_rea
     if (len
```

```
\r\n";
;
```

BEEF

```
n up
 data */
aining;
ufsiz : r->
to_read);
```

Fast forward 11 years... Nginx CVE-2012-2028

- Nginx is found to have an exact same issue!

```
--- 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;
    }
```

```
2302 data:
2303
2304     ctx->state = state;
2305     b->pos = pos;
2306
2307     switch (state) {
2308
2309     case sw_chunk_start:
2310         ctx->length = 3 /* "0" LF LF */;
2311         break;
2312     case sw_chunk_size:
2313         ctx->length = 1 /* LF */
2314                       + (ctx->size ? ctx->size + 4 /* LF "0" LF LF */
2315                                    : 1 /* LF */);
2316         break;
2317     case sw_chunk_extension:
2318     case sw_chunk_extension_almost_done:
2319         ctx->length = 1 /* LF */ + ctx->size + 4 /* LF "0" LF LF */;
2320         break;
2321     case sw_chunk_data:
2322         ctx->length = ctx->size + 4 /* LF "0" LF LF */;
2323         break;
2324     case sw_after_data:
2325     case sw_after_data_almost_done:
2326         ctx->length = 4 /* LF "0" LF LF */;
2327         break;
2328     case sw_last_chunk_extension:
2329     case sw_last_chunk_extension_almost_done:
2330         ctx->length = 2 /* LF LF */;
2331         break;
2332     case sw_trailer:
2333     case sw_trailer_almost_done:
2334         ctx->length = 1 /* LF */;
2335         break;
2336     case sw_trailer_header:
2337     case sw_trailer_header_almost_done:
2338         ctx->length = 2 /* LF LF */;
2339         break;
2340
2341     }
2342
2343     if (ctx->size < 0 || ctx->length < 0) {
2344         goto invalid;
2345     }
```

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 same bug in Apache

State r (again)

- **ngx_http_**
  - 57 switch
  - 272 singl
  - 2300+ Sl
- States and nixed together, **u**
- Parser com e issue immediate in Apache

For desert: Shellshock!

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

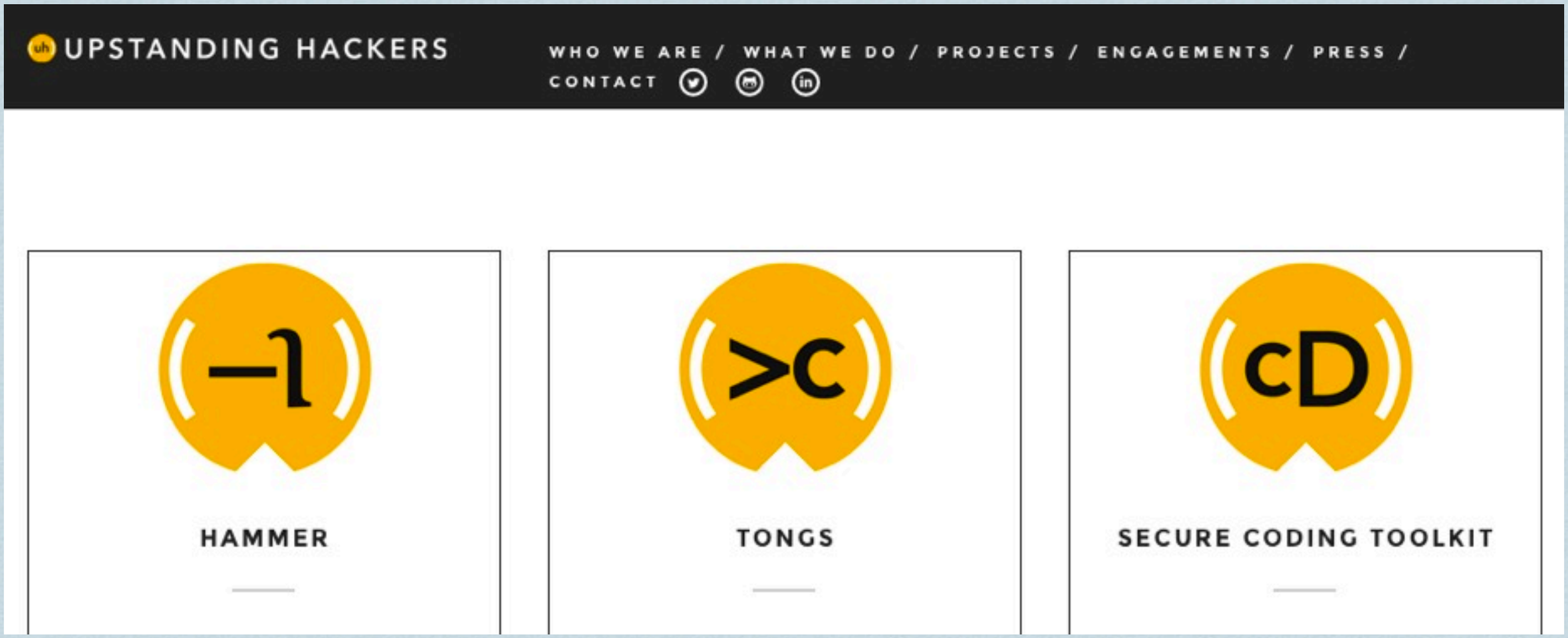
"Bash really is a local app that woke up one morning on the HMS CGI-BIN with a pounding headache"

- **Computation power exposed to external inputs is computation power given to attacker**

For desert: Shellshock!

- 

What future holds



Hammer: https://github.com/abiggerhammer

For parsers that are secure & intelligible

Parser Commandments

- 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
- Don't mix semantic actions with syntax recognition!
  - "Full recognition before processing"
  - Careful with memcopy, etc. before input is fully validated!

LangSec 2015: Join the conspiracy!

How the code auditors describe your software?

http://langsec.org/

May 2015, co-located with IEEE Security & Privacy Symposium

[banners by FX]