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!

Don’t check for voltage with your hand
Кирпич укладывай

Правильно

25

Забо
Кирпич укладывают
Наверху работают
Правильно
25 задов
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*

BruCON 2012, video at langsec.org/
Oldies but goodies

- **BIND 8.4** DNS NXT record overflow, **1999**, by ADM
  - buffer for hostname overflowing; 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”

\texttt{malloc()}

\texttt{memcpy()}

“Input sanitization”

+, -, *, /
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
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
Heartbleed is a parser bug!

Heartbeat sent to victim

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

<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>1 byte</td>
</tr>
</tbody>
</table>
Heartbleed is a parser bug! 😘

Heartbeat sent to victim

SSLv3 record:
- Length

SSL3_RECORD

HeartbeatMessage

<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>1 byte</td>
</tr>
</tbody>
</table>
Heartbleed is a parser bug! ♥️
Heartbleed is a parser bug! 

**Heartbeat sent to victim**

<table>
<thead>
<tr>
<th>SSLv3 record:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
</tr>
<tr>
<td>4 bytes</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>SSLv3 record:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length</td>
</tr>
<tr>
<td>65538 bytes</td>
</tr>
</tbody>
</table>

**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! ♥

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>

Must agree, but never checked

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

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>

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

if (rtype == TLS1 HB REQUEST) {
    unsigned char *buffer, *bp;
    buffer = write_length = 1 /* heartbeat type */ +
    payload + padding;

    if ((buffer = (unsigned char *) malloc(write_length))) {
        // Write heartbeat
        s->msg_write(0, buffer);
        free(buffer);
    }
}

/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rec_length)
    return 0; /* silently discard */

n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rec_length)
    return 0; /* silently discard */

/* Read and process payload */
if (s->msg_callback(0, s->version, TLS1 RT HEARTBEAT, s->s3->rec_data[0], s->s3->rec_length, s->s3->rec_arg, pl = p, n2s(p, payload))
    /* 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->s->rrec.data[0], s->s->rrec.length,
                        s, s->msg_callback_arg);

    /* Read type and payload length first */
    if (1 + 2 + 16 > s->s->rrec.length)
        return 0; /* silently discard */
    hbtype = *p++;
    n2s(p, payload);
    if (1 + 2 + payload + 16 > s->s->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.
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:

FULL RECOGNITION

BEFORE PROCESSING
MANUL THE LANGSEC CAT SAYS:

utf-8 manul by

FULL
RECOGNITION

before processing
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, hand-coded.
- State machine done wrong: code must be generated!
An aside: GnuTLS Hello

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

```python
- 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)
# 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
' ...' # Session ID Data (Payload)

MaliciousServerHello = (num
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
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
Android Master Key: Parser Differentials

Verification

- Unzip
- Verify

Installation

- Unzip
- Install

Bad signature?

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

Turing-complete
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 ";
    # 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

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

```c
--- 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 @@

    len_to_read = (r->remaining > bufsiz) ? bufsiz : r->remaining;
    len_to_read = (r->remaining > (unsigned int)bufsz) ? bufsiz : r->
    remaining;

    len_read = ap_bread(r->connection->client, buffer, len_to_read);
    if (len_read <= 0) {
```
Fast forward 11 years...

Nginx CVE-2012-2028

- 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 same bug in Apache
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

Look under your feet!
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.

- Computation power exposed to external inputs is computation power given to attacker.
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 memcpy, 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]