From “shotgun parsers” to more secure stacks

Sergey Bratus
Meredith L. Patterson
Dan ‘TQ’ Hirsch
“Shotgun parser”, the deadliest of patterns

- Input data checking, handling interspersed with processing logic
Dispatches from the Beagle

- Travel to the past
- Collect specimens of vulns
- Build a cladistics
“Darwin’s Rootshell Finches”

- Complex software written by experts
- Subtle bugs that took a while to find & exploit
- Critical: remote code exec, pre-auth, core protocols/stacks
- Underlying data format complexity reason why bugs happened
A Brief Recap of LangSec

- Recognizer handles input, enforces expectations of subsequent code, paranoid is good.
- Processing code gets the job done, less paranoid (but “might need more sanity checks”).
A Brief Recap of LangSec

- Recognizer handles input, enforces expectations of subsequent code, paranoid is good.
- Processing code gets the job done, less paranoid (but “might need more sanity checks”).
A Brief Recap of LangSec

- Recognizer handles input, enforces expectations of subsequent code, paranoid is good.

- Processing code gets the job done, less paranoid (but “might need more sanity checks”).
A Brief Recap of LangSec

- Recognizer handles input, enforces expectations of subsequent code, paranoid is good.

- Processing code gets the job done, less paranoid (but “might need more sanity checks”).
“Bringing the Wrong Weapon to a Fight”

**Recognizer** is your system’s weapon against programming by crafted input ("weird machines")

- Nearly every **Rails** bug so far, 2013
- IE8 anti-XSS filters fiasco, Pwnie for Most Epic Fail 2010

“Tool-using Finch”
The Lulziest Myths of Input Handling (I)

- **Input sanitization**: “you can suppress ‘bad stuff’ in `input` to make it safe”

- Reality: Safety is a property of your input as a *language*. Only recognition assures it.
The Lulziest Myths of Input Handling (II)

• **Escaping** is “just string replacement”

• Reality: Proper escaping is a *language* property. Only recognition assures it.
The Lulziest Myths of Input Handling (III)

- **Input sanitization**: “you can suppress ‘bad stuff’ in input + output to make it safe”

IE8 Anti-XSS Epic Fail

• IE8 deploys **RegExp** rewriting of server responses to suppress XSS

```html
<OB{J}ECT[ /+\t].*?((type)|(codetype)|(classid)|(code)|(data))[ /+\t]*=

<LI{N}K[ /+\t].*?href[ /+\t]*=

<[i]?f{r}ame.*?[ /+\t]*?src[ /+\t]*=
```

• Renders “safe” sites vulnerable:
  “Abusing IE8s XSS Filters”, Vela Nava & Lindsay, [http://p42.us/ie8xss/](http://p42.us/ie8xss/)

• Google saves:
  `X-XSS-Protection: 0`
“Have substitution, will compute”

- Substitution is computation, too, especially when some component will do it repeatedly for you

- Best ex.: Mario Heiderich’s “Got your Nose”: no-JS CSS-only HTML password recovery
  - password manager brings the loop
  - SVG elements bring the “if”
  - suddenly, it’s a party in your browser
“Ruby off Rails”
CVE-2013-0333
"The problem with the Yaml backend is that its convert_json_to_yaml method is incredibly naive ... [it] uses StringScanner to walk through the JSON string, replacing JSON tokens with their YAML equivalents."

http://ronin-ruby.github.com/blog/2013/01/28/new-rails-poc.html
“The method does not fully parse JSON in order to emit proper YAML...”

- In other words, a finite state transducer
- But we know JSON and YAML are both context-free
- This never had to happen :(
“...nor does it validate that the input is actually valid JSON.”

--- !ruby/hash:ActionController...
“RootShell Finches”

- OpenSSH 3.3 Pre-auth challenge-response, by GOBBLES, 2002
- BIND 8.2 NXT record remote buffer overflow, by ADM [horizon/plaguez], 1999
- OpenBSD 4.0 remote IPv6 mbuf overflow, by Core [ortega, gera], 2007
Your data format is a language. Treat it as such.

- Make elements validatable on their own.
- Avoid having to validate complex relationships between multiple elements ("context sensitivity") in input data.
- The more context you need, the more the devil has you.
OpenSSH 3.3 Pre-Auth remote buffer overflow

• Challenge-response vuln, exploited by GOBBLES (sshutuptheo.tar.gz)

• “Heap-based overflow resulting from an integer overflow”

• Reasonable-looking byte-buffer parser -- but something went awry
“Just us shotgun bytes here”

```c
static void
input_userauth_info_response(int type, u_int32_t seq, void *ctxt)
{
    nresp = packet_get_int();
    if (nresp > 0) {
        response = xmalloc(nresp * sizeof(char*));
        for (i = 0; i < nresp; i++)
            response[i] = packet_get_string(NULL);
    }
    packet_check_eom();
}
```

Consumes 4 bytes off &incoming_packet

Consumes 4 bytes off &incoming_packet, then so many bytes

Aborts packet if trailing bytes
How did it work?

input_userauth_info_response(bad_packet)
xmalloc(nresp * sizeof(char*)) // too-big nresp -> too little memory
packet_get_string(NULL)
buffer_get_string(bad_packet, length_ptr)
buffer_get_int(bad_packet) // len > 256K -> jump to error handling
fatal(...)  
fatal_cleanup()
(*cu->proc)(cu->context) // stomp on *cu->proc and you win!
Where did it go wrong?

- * operator manipulates user-supplied value
- The result of the arithmetic is not checked!
- Not enough memory allocated
- Read \textit{nresp} strings ... all copies of shellcode
  - Since buffer’s too small, stomps memory
  - Specifically, (cu->proc)
- Read too-long string --> fatal()
- Pointer to cu->proc dereferenced == you win!
How to do it right:

```c
h_sequence(h_length_value(h_uint32(),
    h_length_value(h_uint32(),
        h_char()),
    h_end_p()));
```
The syntax-semantics boundary is a boundary of competence

- “Special cases” in code are either features of the input data language -- and must be treated as such -- or are violations of syntax-semantics boundary, and should be avoided

- “Code smells” may signal problems with data design, or worse.
BIND 8.2 ADM-NXT
remote buffer overflow

- Representing a definite negative is hard

- **NXT**: Signable DNS record type containing the interval containing a non-existent name:
  

- Added in RFC 2065, updated by RFC 2535
NXT query scheme

+------------------+  +----------+  +------------------+
| Target Nameserver|  <-------| Exploited Master NS |
|                  |  |           |                      |
| ^                |  |           |                      |
| -1-              |  |           |                      |
|                  |  <-+         |                      |
| +------------------+  +----------+  +------------------+
| Joe Random Hacker |  +---4-----+  | Exploit Delivery NS |
|                  |  +----------+  |                      |
| +------------------+  +----------+  +------------------+
|                  |  +----------+  |                      |

t666.c
Recap: DNS & its RR

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What is A record for <a href="http://www.unixwiz.net">www.unixwiz.net</a>?</td>
<td></td>
</tr>
<tr>
<td>unixwiz.net NS = linux.unixwiz.net</td>
<td>2 dy</td>
</tr>
<tr>
<td>unixwiz.net NS = cs.unixwiz.net</td>
<td>2 dy</td>
</tr>
<tr>
<td>linux.unixwiz.net A</td>
<td>64.170.162.98</td>
</tr>
<tr>
<td>cs.unixwiz.net A</td>
<td>8.7.25.94</td>
</tr>
</tbody>
</table>

Glue Records

TTL
“When you have a shotgun parser, Mr. Length Field is no longer your friend”
“Context sensitive is not a safe place to be”

- Domain name is compressed
- Can only be checked after expanded with offsets to substrings in preceding packet
- The expanded length must be consistent/expected by the result buffer
“Oh where did we go wrong...”

1. case T_NXT:
2.     n = dn_expand(msg, eom, cp, (char *)data, sizeof data);
3.     if (n < 0) {
4.         hp->rcode = FORMERR;
5.         return (-1);
6.     }
7.     if (!ns_nameok((char *)data, class, NULL, response_trans,
8.         domain_ctx, dname, from.sin_addr)) {
9.         hp->rcode = FORMERR;
10.        return (-1);
11.     }
12.     cp += n;
13.     cpl = data + strlen((char *)data) + 1;
14.     memcpy(cpl, cp, dlen - n);
15.     cp += (dlen - n);
16.     cpl += (dlen - n);
17.     /* compute size of data */
18.     n = cpl - (u_char *)data;
19.     cpl = (u_char *)data;
20.     break;
Beware of context-sensitive data formats

• Elements that must add up across a span of data are danger

• “I’ll go parsing until the packet makes sense, then discard the allocs” is danger

• The more context you need, the more the devil has you.
const HParse* init_character_string() {
    static const HParse *cstr = NULL;
    if (cstr)
        return cstr;

    cstr = h_length_value(h_uint8(), h_uint8());
    return cstr;
}

H_ARULE(cstr, init_character_string());
H_ARULE(txt, h_many1(cstr));
const HParsedToken *act_txt(const HParseResult *p) {
  dns_rr_txt_t *txt = H_ALLOC(dns_rr_txt_t);
  const HCountedArray *arr = H_CAST_SEQ(p->ast);
  uint8_t **ret = h_arena_malloc(arr->arena, sizeof(uint8_t*)*arr->used);
  for (size_t i=0; i<arr->used; ++i) {
    size_t len = h_seq_len(arr->elements[i]);
    uint8_t *tmp = h_arena_malloc(arr->arena, sizeof(uint8_t)*len);
    for (size_t j=0; j<len; ++j)
      tmp[j] = H_INDEX_UINT(arr->elements[i], j);
    ret[i] = tmp;
  }
  txt->count = arr->used;
  txt->txt_data = ret;
  return H_MAKE(dns_rr_txt_t, txt);
}
OpenBSD 4.0 remote kernel mbuf overflow

• Found by Core’s ortega, gera Apr ’07

• Kernel remote exploitable IPv6 buffer overflow via ICMPv6 **fragmentation**

• Interacts complexly with **mbuf** packet buffer allocation scheme of OpenBSD
IPv6

<table>
<thead>
<tr>
<th>IPv4 Header</th>
<th>IPv6 Header</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>Version</td>
</tr>
<tr>
<td>IHL</td>
<td>Traffic Class</td>
</tr>
<tr>
<td>Type of</td>
<td>Flow Label</td>
</tr>
<tr>
<td>Service</td>
<td></td>
</tr>
<tr>
<td>Total Length</td>
<td></td>
</tr>
<tr>
<td>Identification</td>
<td></td>
</tr>
<tr>
<td>Flags</td>
<td></td>
</tr>
<tr>
<td>Fragment Offset</td>
<td></td>
</tr>
<tr>
<td>Time to Live</td>
<td></td>
</tr>
<tr>
<td>Protocol</td>
<td></td>
</tr>
<tr>
<td>Header Checksum</td>
<td></td>
</tr>
<tr>
<td>Source Address</td>
<td></td>
</tr>
<tr>
<td>Destination Address</td>
<td></td>
</tr>
<tr>
<td>Options</td>
<td></td>
</tr>
<tr>
<td>Padding</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Next Header

IPv6 Header -> Upper Layer Header -> Data

Next Header

IPv6 Header -> Extension Header

Next Header

IPv6 Header -> Extension Header

Next Header

IPv6 Header -> Extension Header

Next Header

IPv6 Header -> Upper Layer Header -> Data
Chaining headers by NH type

[Diagram showing the structure of packet headers with chaining headers by NH type.]
struct m_hdr {
    struct mbuf *mh_next;
    struct mbuf *mh_nextpkt;
    caddr_t mh_data;
    u_int mh_len;
    short mh_type;
    u_short mh_flags;
};

struct pkthdr {
    struct ifnet *rcvif;
    SLIST_HEAD(packet_tags, m_tag) tags;
    int len;
    int csum_flags;
    struct pkthdr_pf;
};
What happens with mbufs

- Packets are stored in chains of mbufs
- Headers get parsed & turned into memory representation one at a time
- mbufs get copied and changed in place, depending on previous mbufs in the chain
- Very context-sensitive
Ensuring mbuf bytes are contiguous in memory

```c
/*
 * ensure that [off, off + len) is contiguous on the mbuf chain "m".
 * packet chain before "off" is kept untouched.
 * if offp == NULL, the target will start at <retval, 0> on resulting chain.
 * if offp != NULL, the target will start at <retval, *offp> on resulting chain.
 * on error return (NULL return value), original "m" will be freed.
 * XXX M_TRAILINGSPACE/M.LEADINGSPACE on shared cluster (sharedcluster)
 */
struct mbuf *
m_pulldown(struct mbuf *m, int off, int len, int *offp)
{
    struct mbuf *n, *o;
    int hlen, tlen, olen;
    int sharedcluster;

    /* check invalid arguments. */
    if (m == NULL)
        panic("m == NULL in m_pulldown()");
    if (len > MCLBYTES) {
        m_freem(m);
        return (NULL); /* impossible */
    }
```
/* when len <= n->m_len - off and off != 0, it is a special case.
* len bytes from <n, off> sits in single mbuf, but the caller does
* not like the starting position (off).
* chop the current mbuf into two pieces, set off to 0.
*/
if (len <= n->m_len - off) {
    struct mbuf *mlast;

    o = m_dup1(n, off, n->m_len - off, M_DONTWAIT);
    if (o == NULL) {
        m_freem(m);
        return (NULL); /* ENOBUFS */
    }

static struct mbuf *
_m_dup1(struct mbuf *m, int off, int len, int wait)
{
    struct mbuf *n;
    int l;
    int copyhdr;

    if (len > MCLBYTES)
        return (NULL);
    if (off == 0 && (m->m_flags & M_PKTHDR) != 0) {
        copyhdr = 1;
        MGETHDR(n, wait, m->m_type);
        l = MHLEN; /* SB: 256 - m_hdr - pkthdr */

        m_copydata(m, off, len, mtod(n, caddr_t));
void
ip6_input(m)
{
    struct mbuf *m;

    struct ip6_hdr *ip6;
    int off = sizeof(struct ip6_hdr), nest;
    u_int32_t plen;
    u_int32_t rtalert = ~0;
    int nxt, ours = 0;
    struct ifnet *deliverifp = NULL;

    /* 451 lines omitted */

    /*
     * protection against faulty packet - there should be
     * more sanity checks in header chain processing.
     */
    if (m->m_pkthdr.len < off) {
        ip6stat.ip6s_tooshort++;
        in6_ifstat_inc(m->m_pkthdr.rcvif, ifs6_in_truncated);
        goto bad;
    }

    bad: /*inet6sw[ip6_protox[nxt]].pr_input)(&m, &off, nxt);
What does this code smell like?

• Code smells are hints, not certainties
• Pragmatism dictates: look deeper.

Printed on fan-fold paper, no function should be longer than you are tall!
This is often a symptom of violating the OneResponsibilityRule.

One Responsibility Rule

From BertrandMeyer's ObjectOrientedSoftwareConstruction, there was the statement (quoting from memory):

A class has a single responsibility: it does it all, does it well, and does it only.

When a function has too many responsibilities, it becomes buried deep in SpecialFormatting, which has a CodeSmell.

To avoid bloat and confusion, and ensure that code is truly simple (not just quick to hack out) we have to practice CodeNormalization, which seems to be a variation on OnceAndOnlyOnce and also DoTheSimplestThingThatCouldPossiblyWork.
What were they trying to do?

Composed Method

Keep all of the operations in a method at the same level of abstraction.

- You’d think one layer of the network stack would be one layer of abstraction
- But its syntax and semantics are different layers
Design has been about code patterns; it should also be about data patterns

• What do we mean when we say “offset”?
What We Talk About When We Talk About Offsets

• Packet offsets
  • Where in the packet does data start?

• Buffer offsets
  • Where in the buffer does an item start?

• 1 packet == 1 buffer == same value
  • otherwise, not necessarily!
How did they fix it?

```c
@@ -226,16 +226,16 @@ m_dup1(struct mbuf *m, int off, int len,
 {
     struct mbuf *n;
     int l;
-    int copyhdr;
+
     if (len > MCLBYTES)
         return (NULL);
     if (off == 0 && (m->m_flags & M_PKTHDR) != 0) {
-        copyhdr = 1;
-        MGETHDR(n, wait, m->m_type);
+        if (n == NULL)
+            return (NULL);
        M_DUP_PKTHDR(n, m);
        l = MHLEN;
     } else {
-        copyhdr = 0;
-        MGET(n, wait, m->m_type);
         l = MLEN;
     }

@@ -249,8 +249,6 @@ m_dup1(struct mbuf *m, int off, int len,
 if (!n)
     return (NULL);
-
     if (copyhdr)
         M_DUP_PKTHDR(n, m);
     m_copydata(m, off, len, mtod(n, caddr_t));
     n->m_len = len;
```
What does this code smell like?

Arrow Anti Pattern

Consists of code where nested if statements generate an arrow shape, like so:

```c
if
  if
    if
      if
        do something
      endif
    endif
  endif
endif
endif
endif
endif
```
@@ -226,16 +226,16 @@ m_dup1(struct mbuf *m, int off, int len,
{
    struct mbuf *n;
    int l;
    int copyhdr;

    if (len > MCLBYTES)
        return (NULL);
    if (off == 0 && (m->m_flags & M_PKTHDR) != 0) {
-        copyhdr = 1;
+        copyhdr = 0;
        MGETHDR(n, wait, m->m_type);
        if (n == NULL)
+            return (NULL);
            return (NULL);
+        M_DUP_PKTHDR(n, m);
        l = MHLEN;
    } else {
-        copyhdr = 0;
+        copyhdr = 1;
        MGET(n, wait, m->m_type);
        l = MLEN;
    }
@@ -249,8 +249,6 @@ m_dup1(struct mbuf *m, int off, int len,
    if (!n)
        return (NULL);

-    if (copyhdr)
-        M_DUP_PKTHDR(n, m);
m_copydata(m, off, len, mtod(n, caddr_t));
n->m_len = len;
You expected magic bullets?

- OSVDB 23199: The TENEX Password Bug
- TENEX?
- Disclosure date: 1972-01-01

Vampire Finch
```
// Kernel mode
// Note: originally written in PDP-10 assembly
int check_password(const char* user, const char* pwd) {
    const char* real_pwd = get_user_password(user);
    int i;
    for (i=0; pwd[i] != 0 && real_pwd[i] != 0; i++)
        if (pwd[i] != real_pwd[i])
            return 0;
    return 1;
}
```
Take-away

• Your data format is a language. Treat it as such.
• Beware of context-sensitive data formats
• Design has been about code patterns; it should be also about data patterns - actually, data languages

“The syntax-semantics boundary is a boundary of competence.”
Protect it with correct recognizers.