



# Shotgun parsers in the cross-hairs

Sergey Bratus  
Meredith L. Patterson



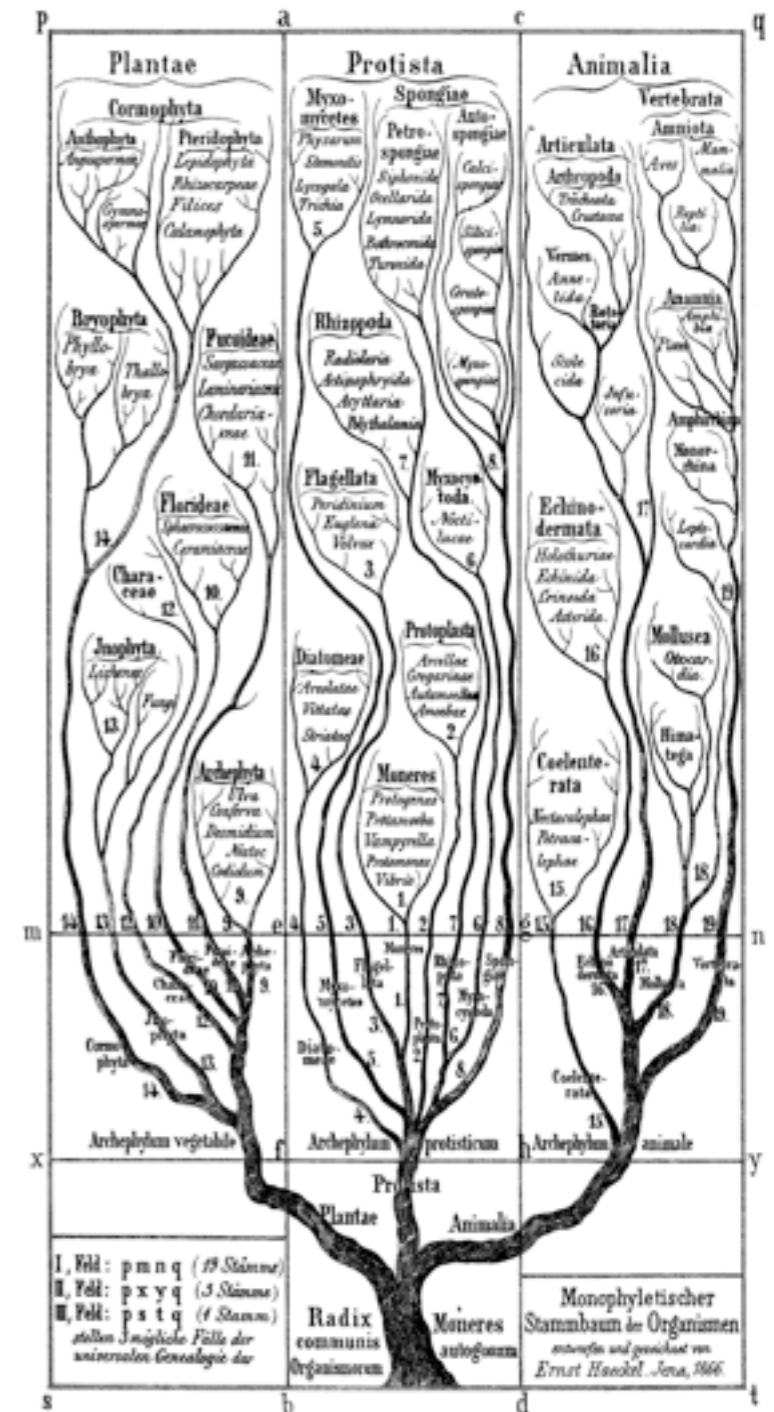
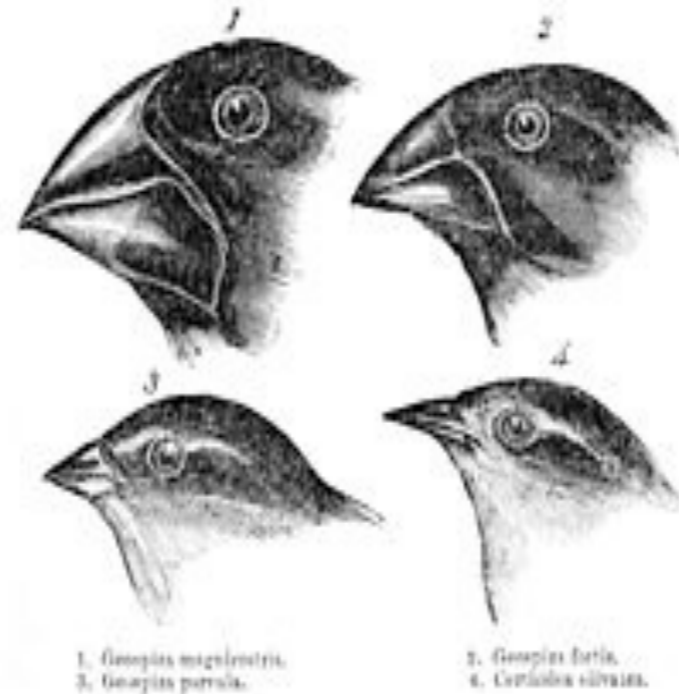
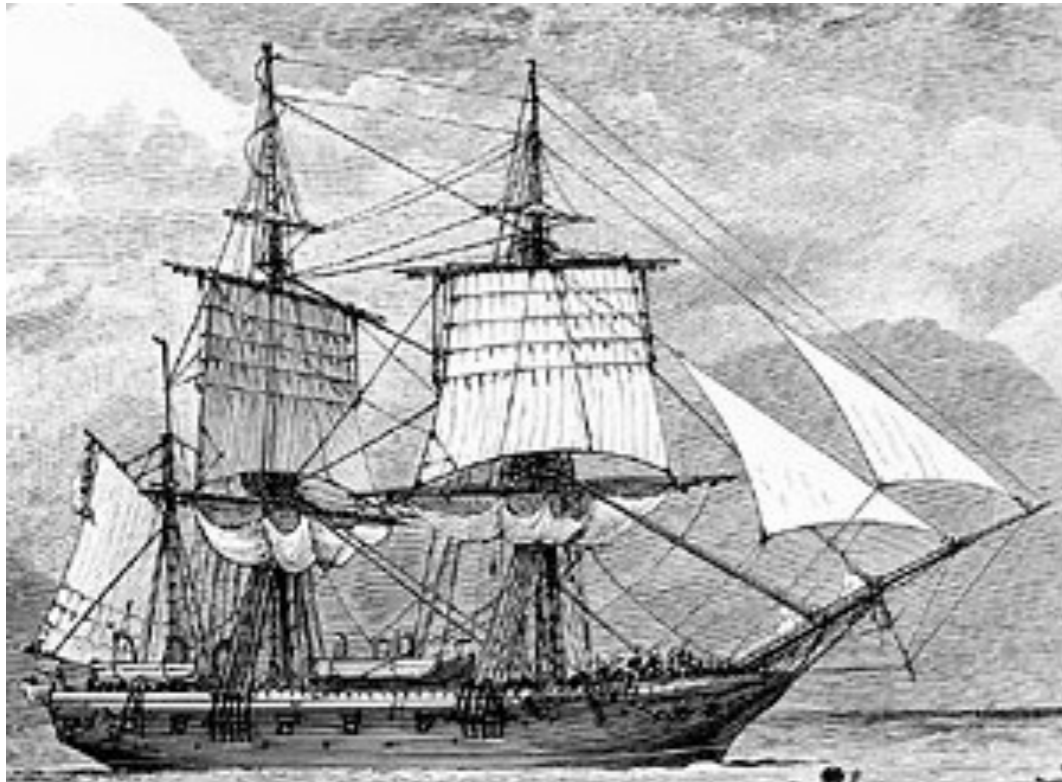
# “Shotgun parser”, the deadliest of patterns

- Input data checking, handling interspersed with processing logic



# Dispatches from the Beagle

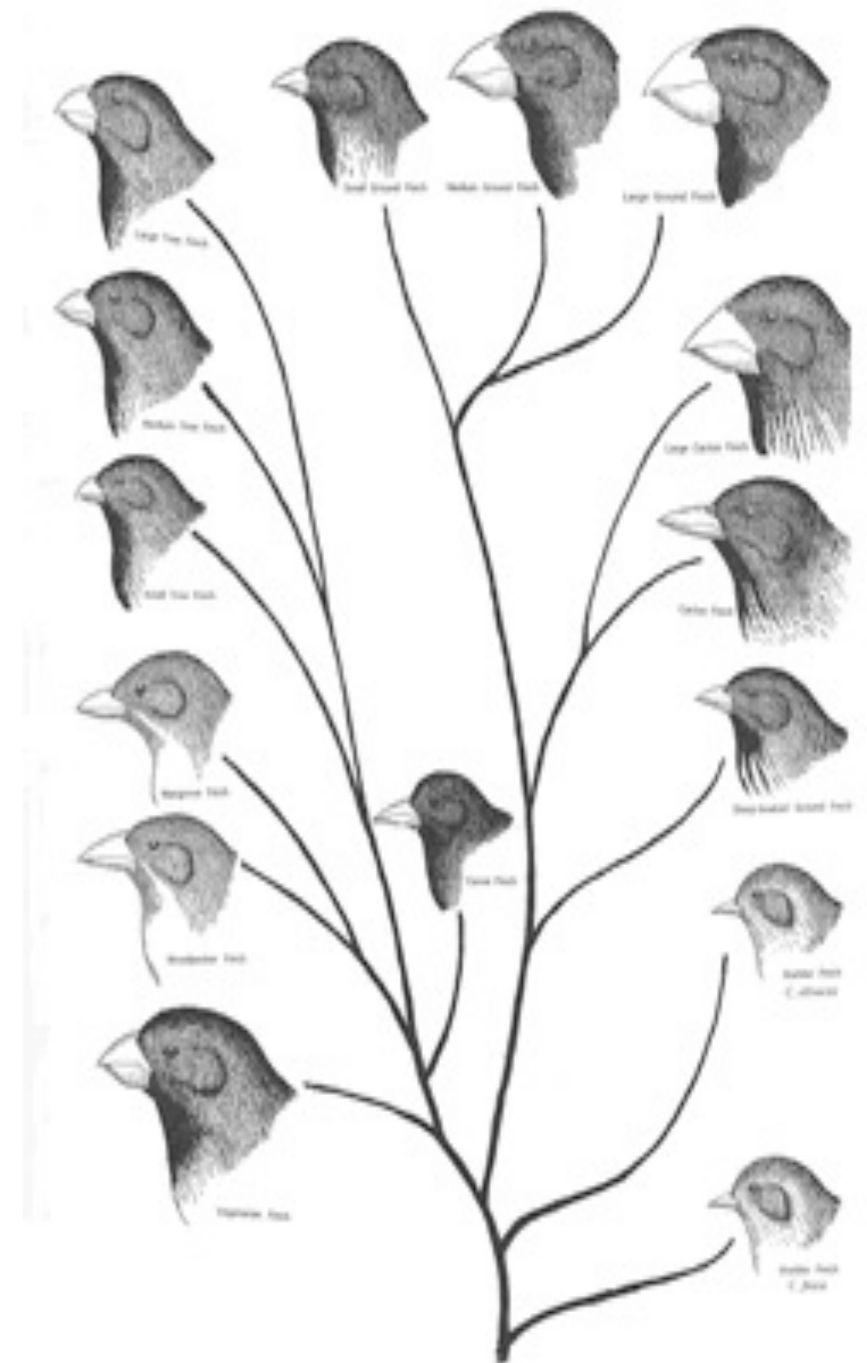
- Travel to the past
- Collect specimens of vulns
- Build a cladistics



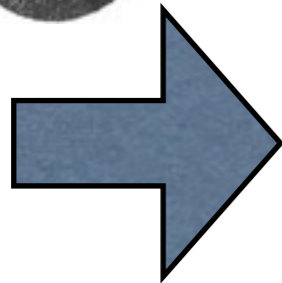


# “Darwin’s Rootshell Finches”

- 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



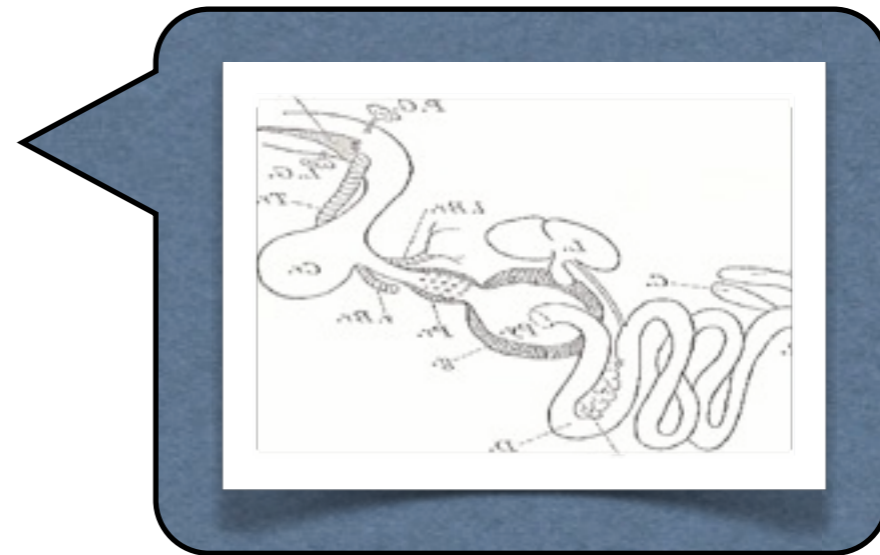
Input



Recognizer



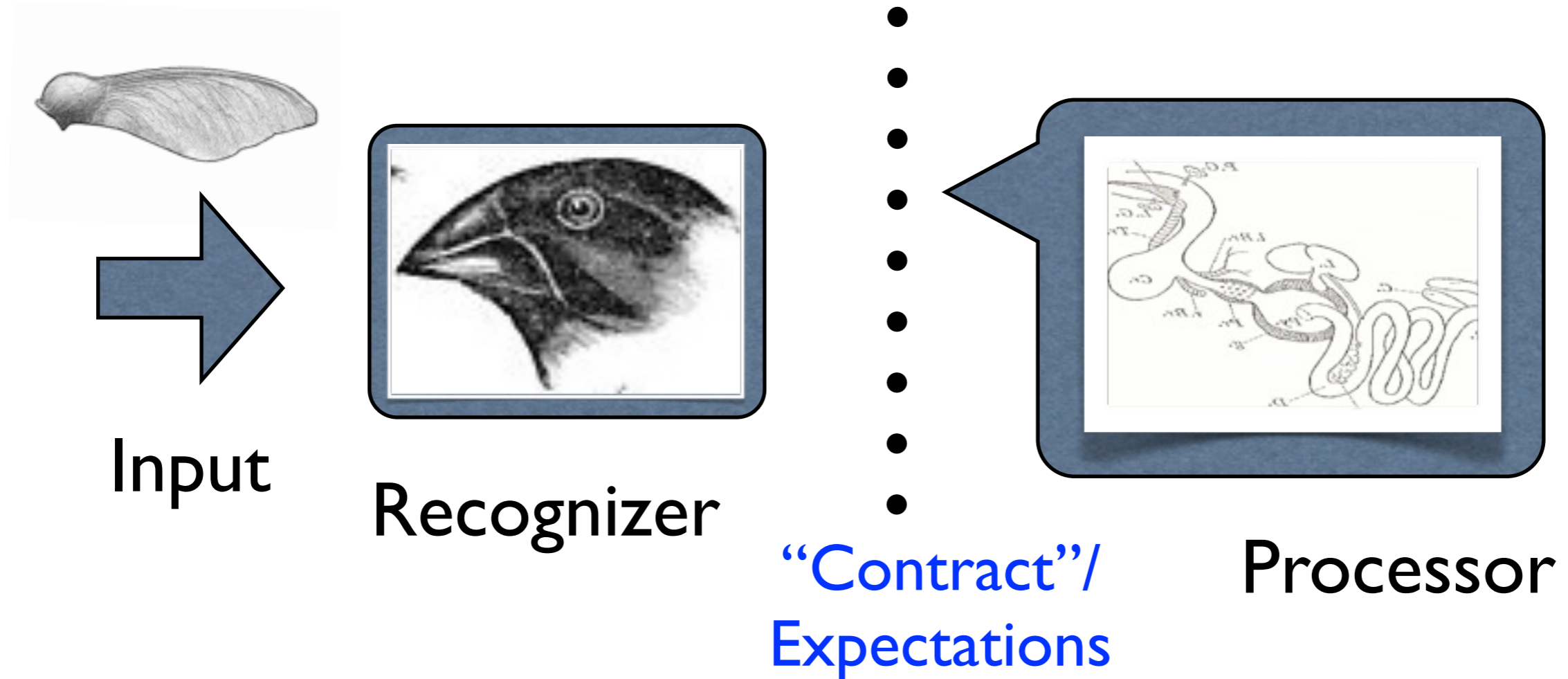
“Contract”/  
Expectations



Processor

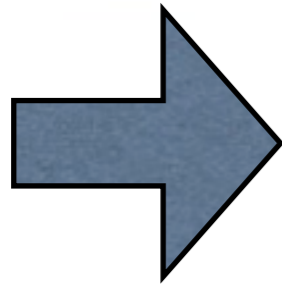
- Recognizer handles input, enforces expectations of subsequent code, paranoid is good.
- Processing code gets the job done, less paranoid (but “might need more sanity checks”).

# 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



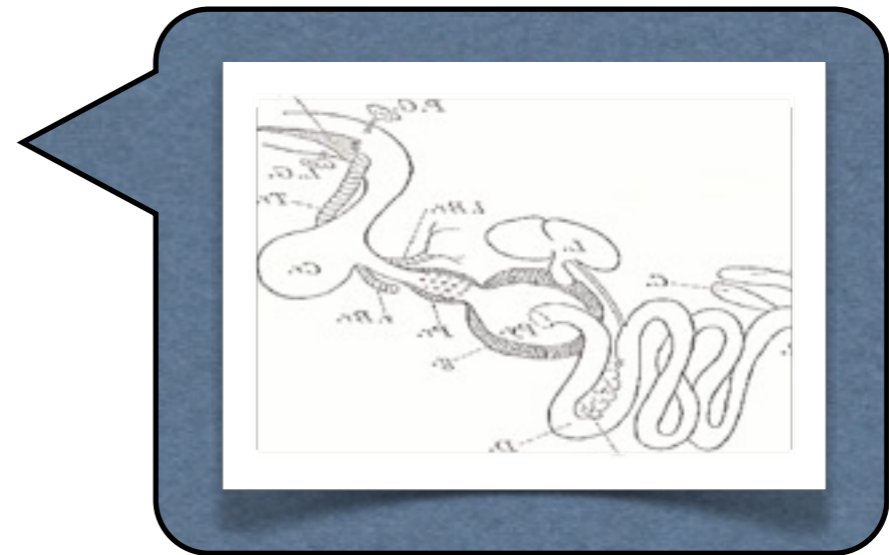
Input



Recognizer



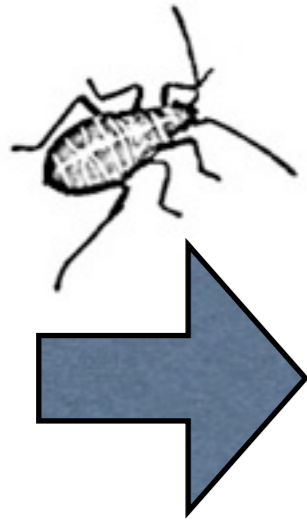
“Contract”/  
Expectations



Processor

- 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



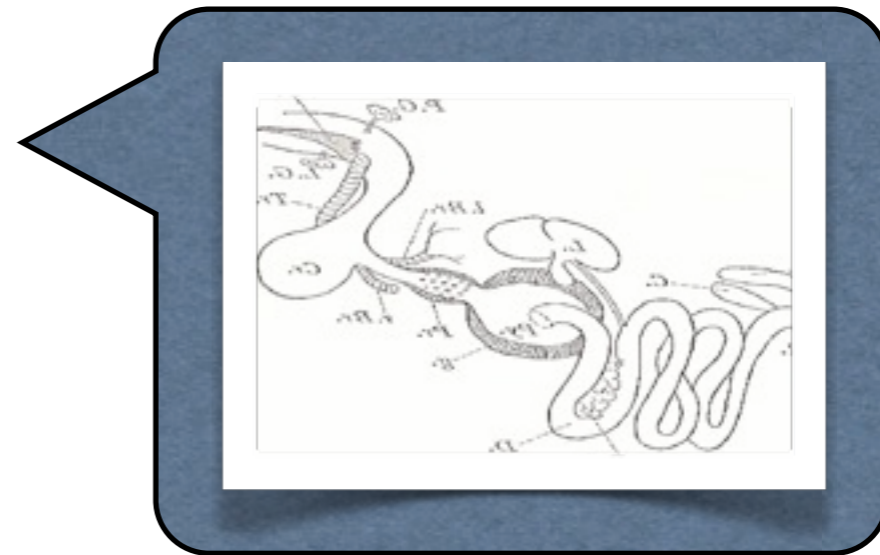
Input



Recognizer



“Contract”/  
Expectations



Processor

- 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”)



- The Reddit Comment Bomb, 2009
- IE8 anti-XSS filters fiasco,  
Pwnie for Most Epic Fail 2010



“Tool-using Finch”

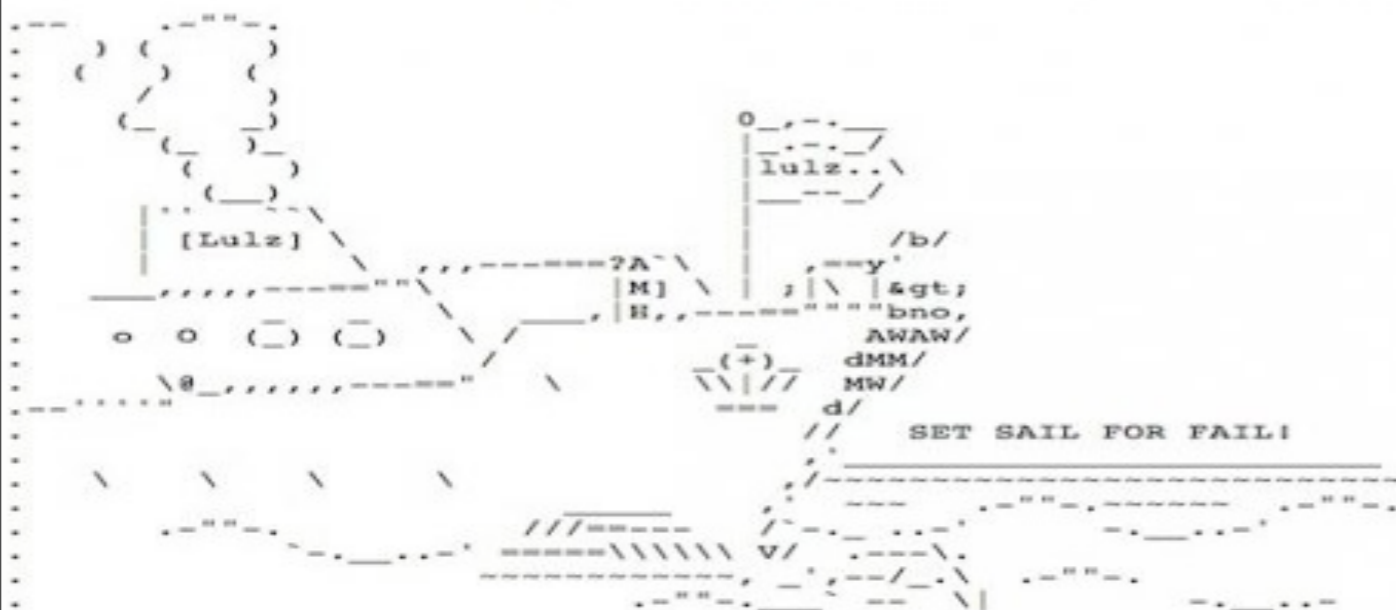
# The Lulziest Myths of Input Handling (I)

- **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.



# Reddit Comment Bomb

—“Reddit weird machine”

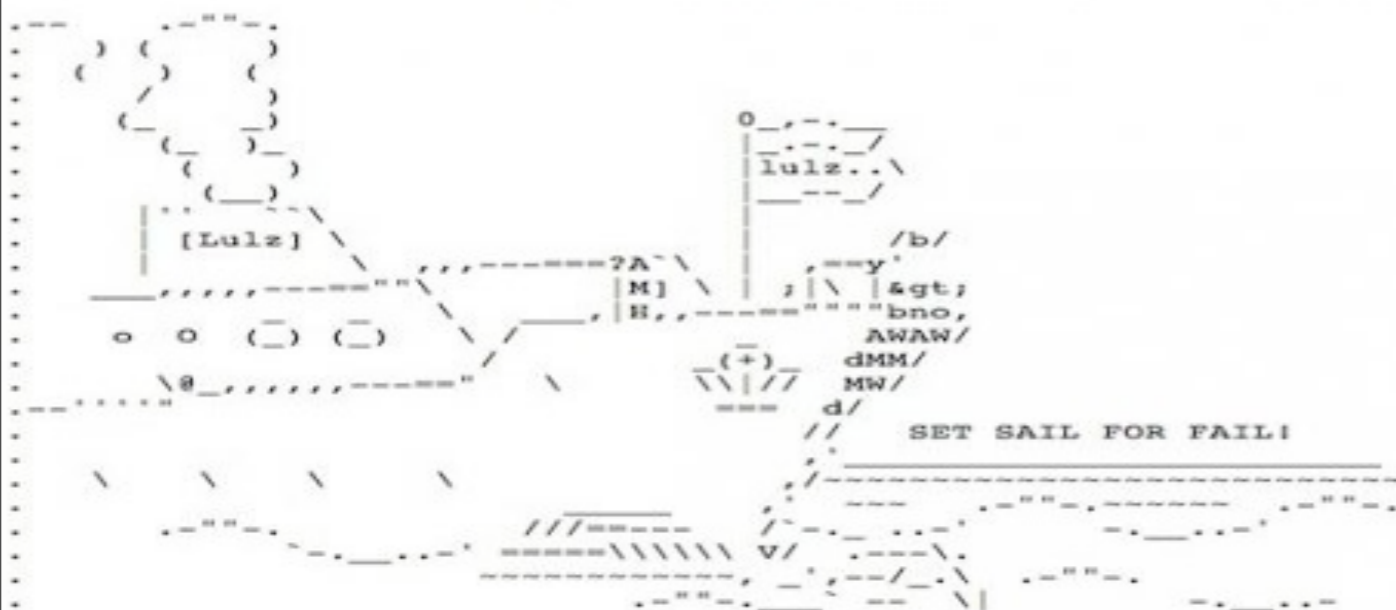
- “to prevent double escaping of certain chars, they are run through MD5 after being escaped once, then MD5 is undone at the end”
- “markdown allowed specifying a variable for inflating later on”
- Create a comment with JS on *onmouseover* injecting *MD5 =>* automatically post comments on user’s behalf



<http://blog.reddit.com/2009/09/we-had-some-bugs-and-it-hurt-us.html>

# The Lulziest Myths of Input Handling (III)

- **Input sanitization:** “you can suppress ‘bad stuff’ in *input+output* to make it safe”
- **Reality:** Halting problem. Deal with it.



# IE8 Anti-XSS Epic Fail



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

```
<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/>

- Google saves:

```
X-XSS-Protection: 0
```

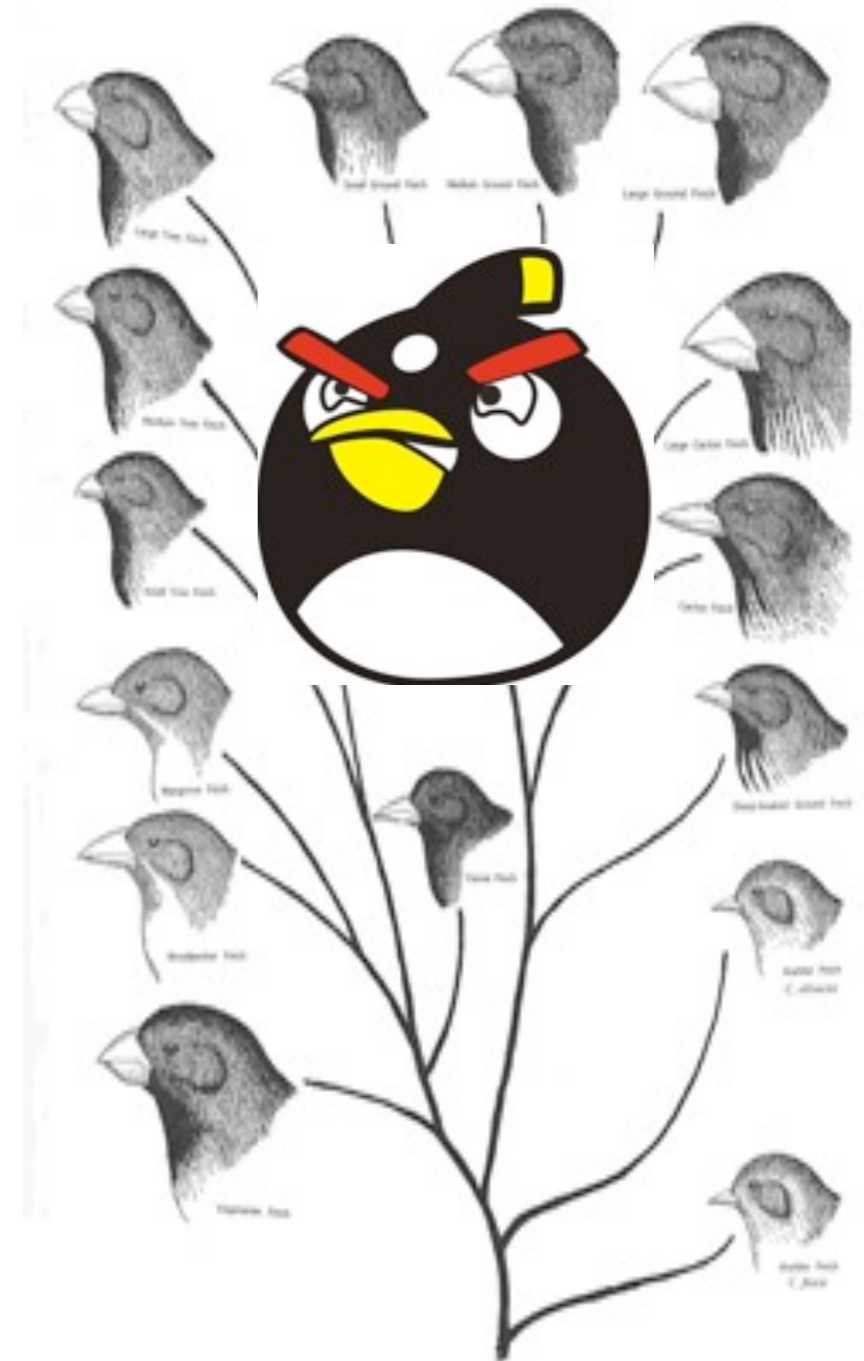
# “Have substitution, will compute”



- Substitution is **computation**, too, especially when some component will do it **repeatedly** for you
- Best ex.: Mario Heiderich’s “*Got your Nose*”: no-JS CCS-only HTML password recovery
  - password manager brings the loop
  - SVG elements bring the “if”
  - suddenly, it’s a party in your browser

# “RootShell Finches”

- BIND 8.2 NXT record remote buffer overflow, by ADM [horizon/plaguez], 1999
- OpenSSH 3.3 Pre-auth challenge-response, by GOBBLES, 2002
- 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.

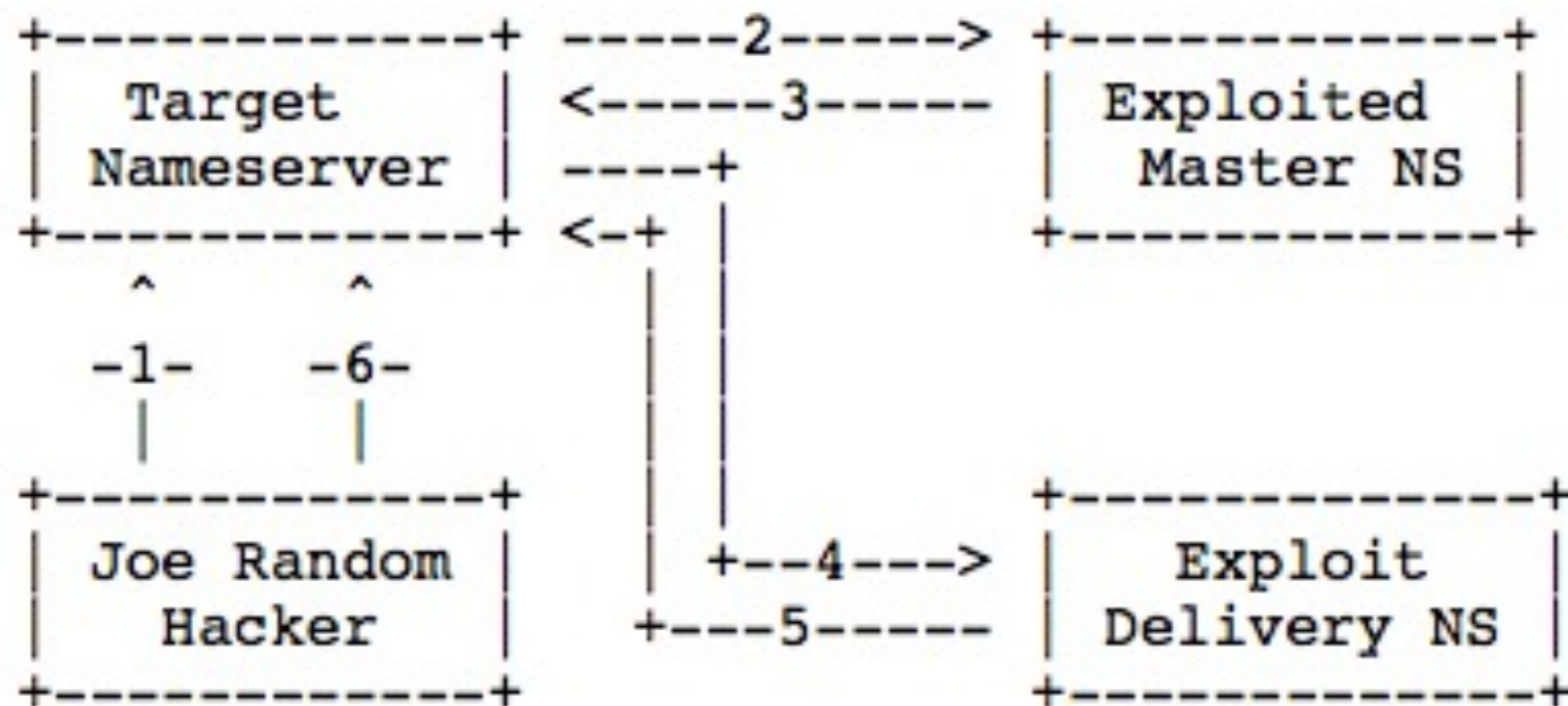
# BIND 8.2 ADM-NXT remote buffer overflow



1999

- Representing a definite negative is hard
- **NXT**: Signable DNS record type containing the interval containing a non-existent name:  
a.foo.com. NXT c.foo.com.
- Added in RFC 2065, updated by RFC 2535

# NXT query scheme



t666.c

# Recap: DNS & its RRs

QID = 43561		1	Op=0	0	T	R	0	Z	rc=ok	
Question count = 1		Answer count = 0								
Authority count = 2		Addl. Record count=2								
Qu	What is A record for www.unixwiz.net?									
Au	unixwiz.net NS =	linux.unixwiz.net					2 dy			
Au	unixwiz.net NS =	cs.unixwiz.net					2 dy			
Ad	linux.unixwiz.net	A = 64.170.162.98					1 hr			
Ad	cs.unixwiz.net	A = 8.7.25.94					1 hr			

Glue Records

TTL

RA=

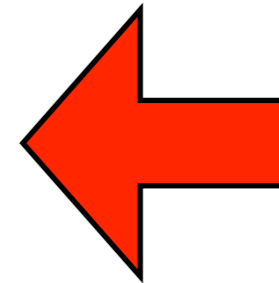


# “Context sensitive is not a safe place to be”

- 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.    cp1 = data + strlen((char *)data) + 1;
14.    memcpy(cp1, cp, dlen - n);
15.
16.    cp += (dlen - n);
17.    cp1 += (dlen - n);
18.
19.    /* compute size of data */
20.    n = cp1 - (u_char *)data;
21.    cp1 = (u_char *)data;
22.    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.



# OpenSSH 3.3 Pre-Auth remote buffer overflow



2002

- Challenge-response vuln, exploited by GOBBLES ([sshutuptheo.tar.gz](http://sshutuptheo.tar.gz))
- “Heap-based overflow resulting from an integer overflow”
- Reasonable-looking byte-buffer parser  
-- but something went awry

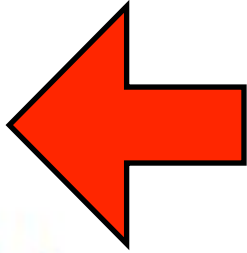
# “Just us shotgun bytes here”

```
static void  
input_userauth_info_response(int type, u_int32_t seq, void *ctxt)  
{
```

Consumes 4 bytes off  
&incoming\_packet



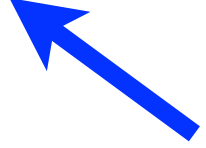
```
    nresp = packet_get_int();  
    if (nresp > 0) {  
        response = xmalloc(nresp * sizeof(char*));  
        for (i = 0; i < nresp; i++)  
            response[i] = packet_get_string(NULL);  
    }  
    packet_check_eom();
```



Aborts packet  
if trailing bytes



Consumes 4 bytes off  
&incoming\_packet,  
then so many bytes



# 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.

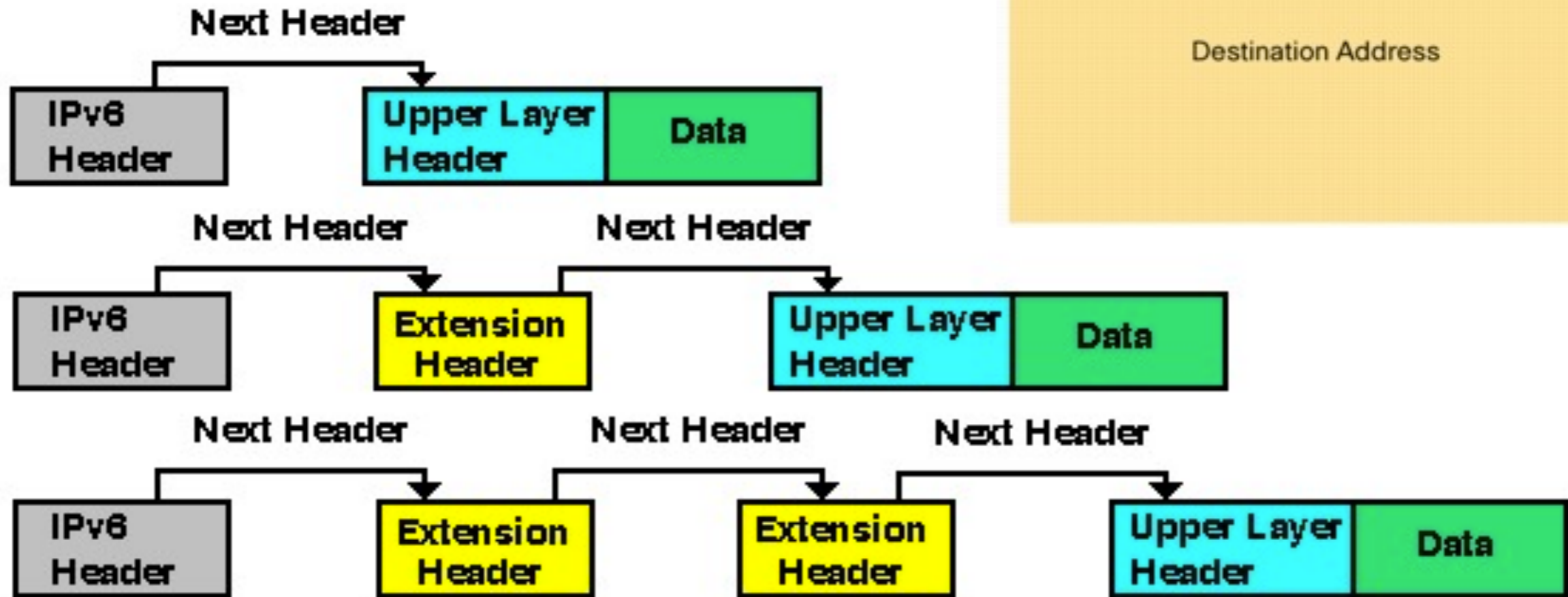
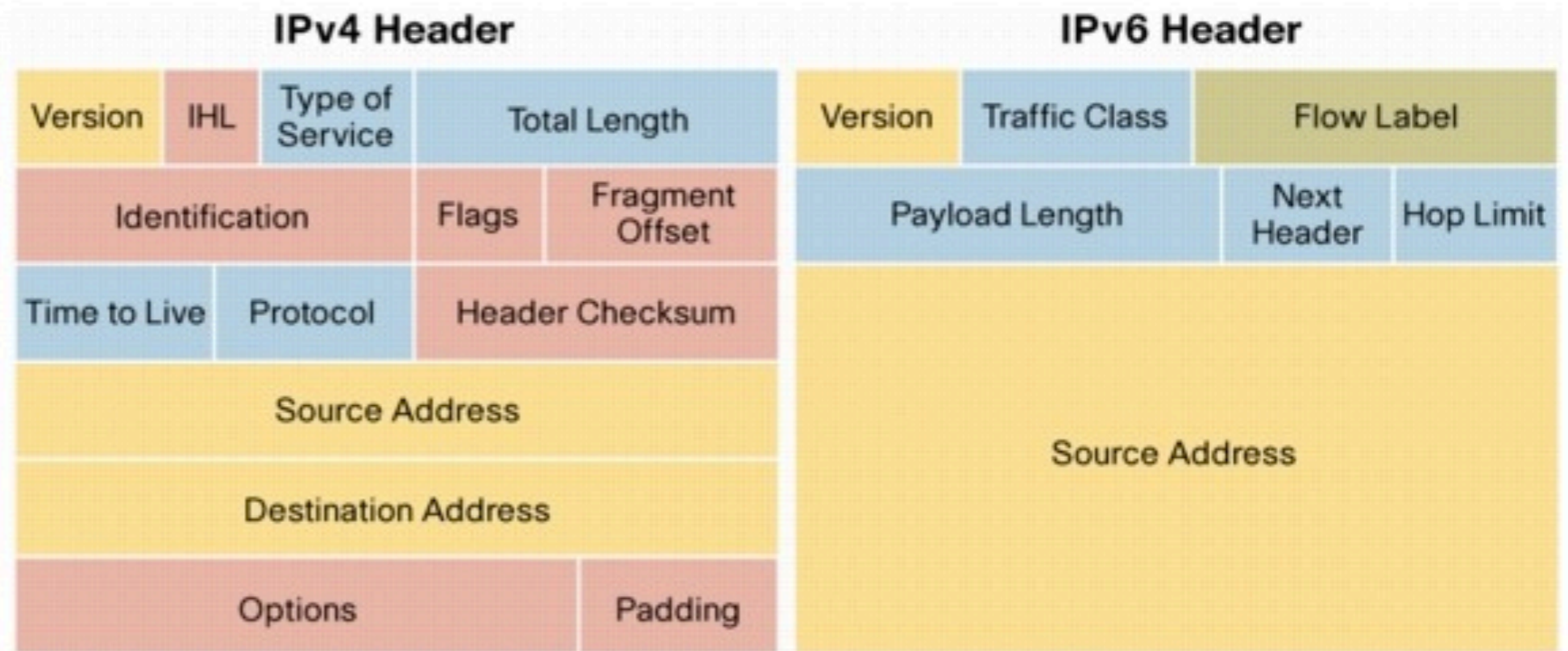
# OpenBSD 4.0 remote kernel mbuf overflow



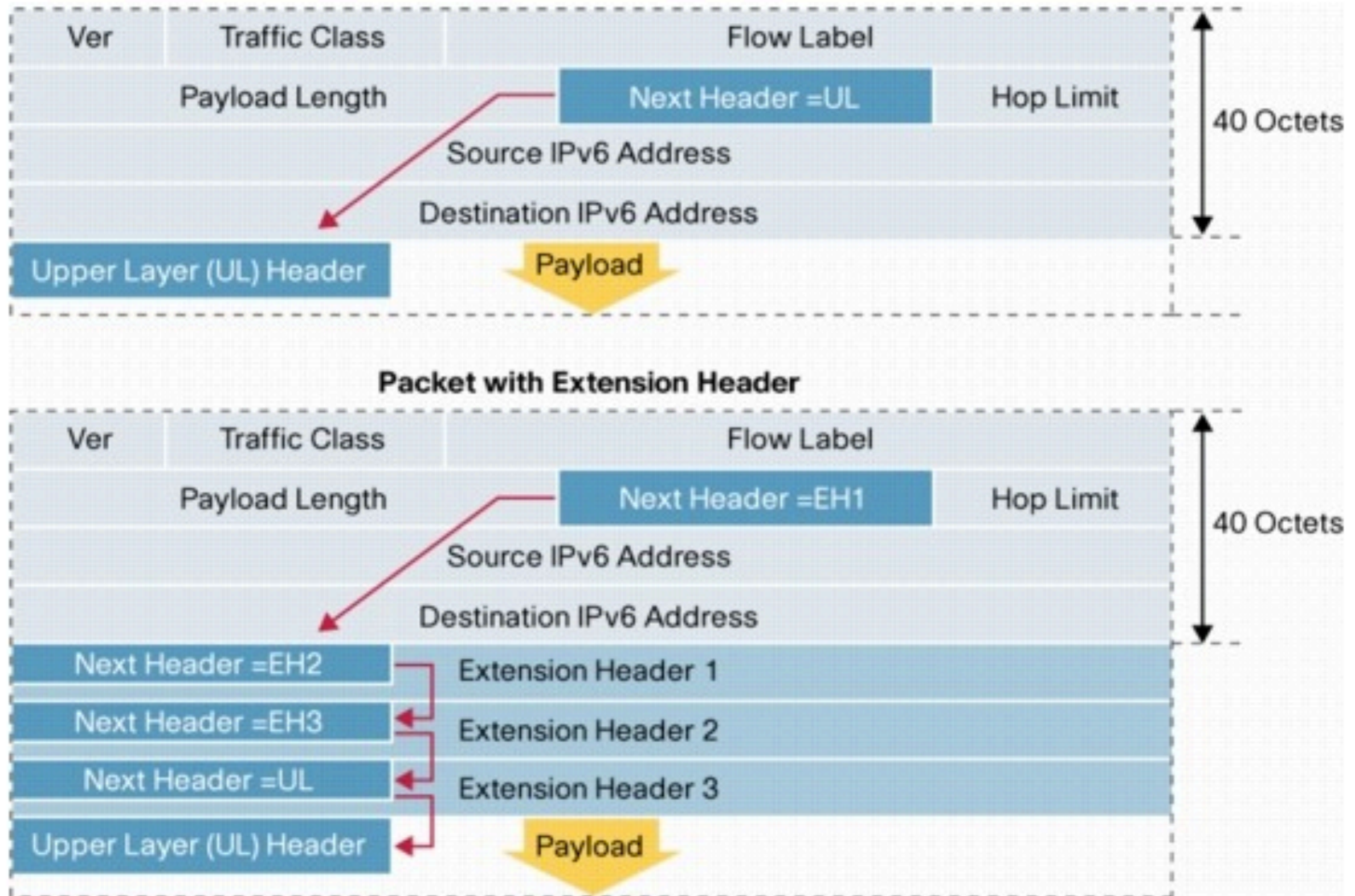
2007

- 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

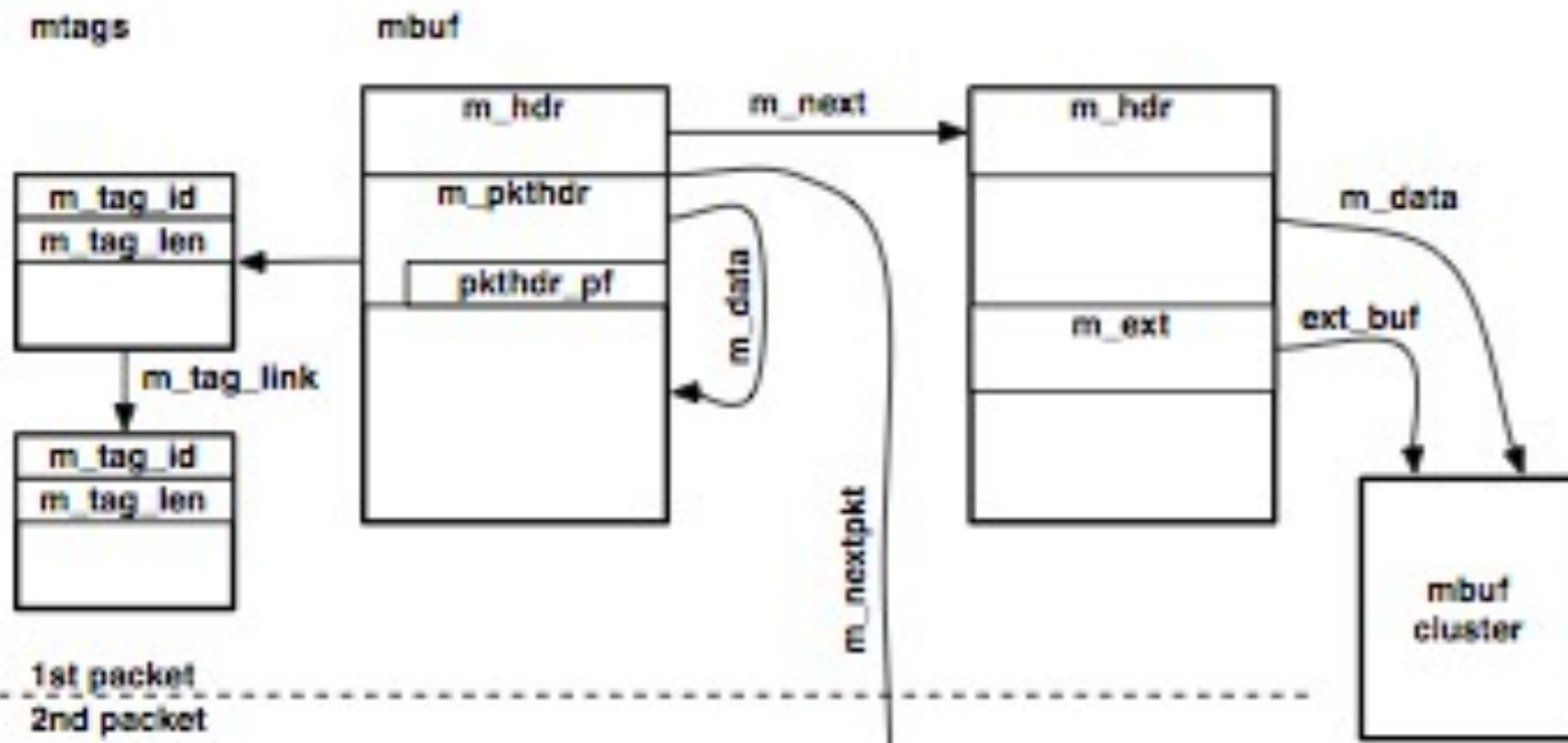
# IPv6



# Chaining headers by NH type



# mbufs



```
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

```
/*
 * 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 rtalet = ~0;
    int nxt, ours = 0;
    struct ifnet *deliverifp = NULL;

```

What does  
this code  
smell like?

/\* 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;
}

```

```

    nxt = (*inet6sw[ip6_protox[nxt]].pr_input>(&m, &off, nxt);

```

```

}
return;

```

```

bad:

```

```

    m_freem(m);

```

```

}

```

# What does this code smell like?



## Taller Than Me

Printed on fan-fold paper, no function should be longer than you are tall!

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

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?
- $| \text{packet} == | \text{buffer} == \text{same value}$ 
  - otherwise, not necessarily!



# How did they fix it?

```
@@ -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:

```
if
  if
    if
      if
        do something
      endif
    endif
  endif
endif
```

# Stinky.

```
@@ -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;
```



# 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.

