FatNS analizes and tests with name servers

mucking about with network security in ruby

Boaz Goldstein and Ohad Luzky

Abstract

You might expect that the abstract would describe the open issue of DNS forensics in the field of security, and how the innovative software we wrote solves it. Thing is, it isn’t, it isn’t and it doesn’t. We just wanted a good grade in Orr’s course, and really like ruby. We will take you on a magical journey full of yields, packet unpacking, evil entities, and for dessert some UI purtyness.

Konichiwa ruby! ^_^

- created by yukihiro matzumoto
- “to make programmers *happy*”
- heavily influenced by perl, eiffel and smalltalk

basic syntax

- like eiffel, but more perl.
- semicolons are optional (linebreaking is intellegent)
- `{ }` and `begin/end`
- `( )` are optional (also, intelligently)

naming conventions

- `normal_variable`
- `@instance_variable`
- `@@class_variable`
- `$evil_perl_global`
- `ConstantsWhichShouldNotChange`
- `|yielded_variable|`
- Classes are Constant. Or are they?

Hashes and symbols

- Hashes are maps, native to ruby.

```
 { 'beach' => 'ball', 'pi' => 3.14159265,
42 => 'unimportant number' }
```

- Symbols are lightweight strings, commonly used in hashes.

```
 { :dhh => 'David Heinmeier Hannson' }
```

Flow control

- No `for` loop. (Hang in there)
- Usual loops, as in perl, with postfix syntax:
  ```
  puts 'Hello' until tired
  ```
- Flow can be yielded back:

```
def give_me_a_t
  yield 'T!'
end

give_me_a_t do |the_t_we_were_expecting|
  puts the_t_we_were_expecting
end
```

Flow control examples

```
1.upto(250000) do |i|
  mail 'amit', "I've told you once if I've",
               "told you #{i} times, ruby",
       "has no for loop!"
end

[ 'some', 'array', 'members' ].each
  { |member| puts member }
```

Objects

```
class Dog
  def initialize
    puts 'Woof woof!'
  end
end

fido = Dog.new

class << fido
  def roll_over
    rotate :axis => gravity.center
  end
end
```

Overview of FatNS

- Designed to sniff DNS communication
- Pluggable attack detection framework
- Purdy GTK2 UI

Demonstration

- Cue the demo ;)

Packet CAPture library

What is pcap

- capture packets
- standard filter format
- standard load/save format
- packet injecion interface
- simple inteface to low level things
  - interfaces
  - promiscous mode
  - blocking/non-blocking (supposedly)
- cross-platform

Pcap-ruby

- provides an infinite-loop style interface
- Privides a blocing get-N-packets interface
- provides an *estimate* of how many packets captured
- file loading/saving, iinterface open/close
- filtering
- *now* lets you see all devices
- pcaplet
- some pre-defined packet formats (IP,TCP,UDP,ICMP)

the blocking Pcap problem

three options exist to get around the block

- Hack the bindings
- Raw sockets
- threads/proccesses

Hack bindings

- pro: seems simple
- pro: self evident
- pro: makes crappyness someone else’s fault
- con: isn’t simple
- con: doesn’t allways work

Raw sockets

- pro: i trust my own code better
- pro: easy to make non-blocking
- con: requiers bindings
- con: linux 2.6 only (2.4 has packet-sockets)
- con: long to write

threads/procceses

- pro: it works
- pro: simple in ruby
- con: buggy synchronized queue
- con: scarry
- con: messy to debug

The internet!!!

DNS may travel over

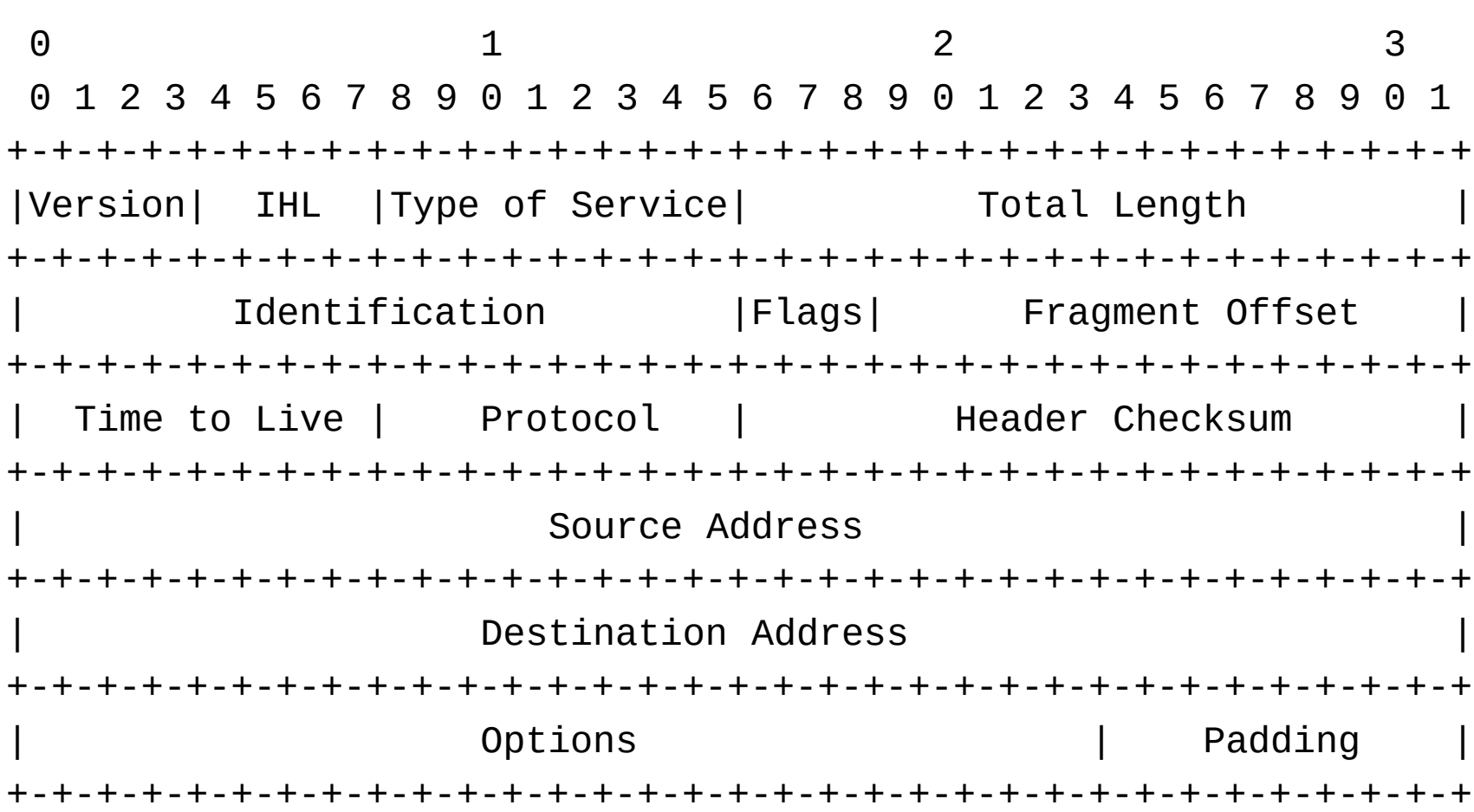
- TCP
- UDP

which may travel over

- IPv4
- IPv6
- there has to be something else too...

we will take care of IP,TCP and UDP.

IP

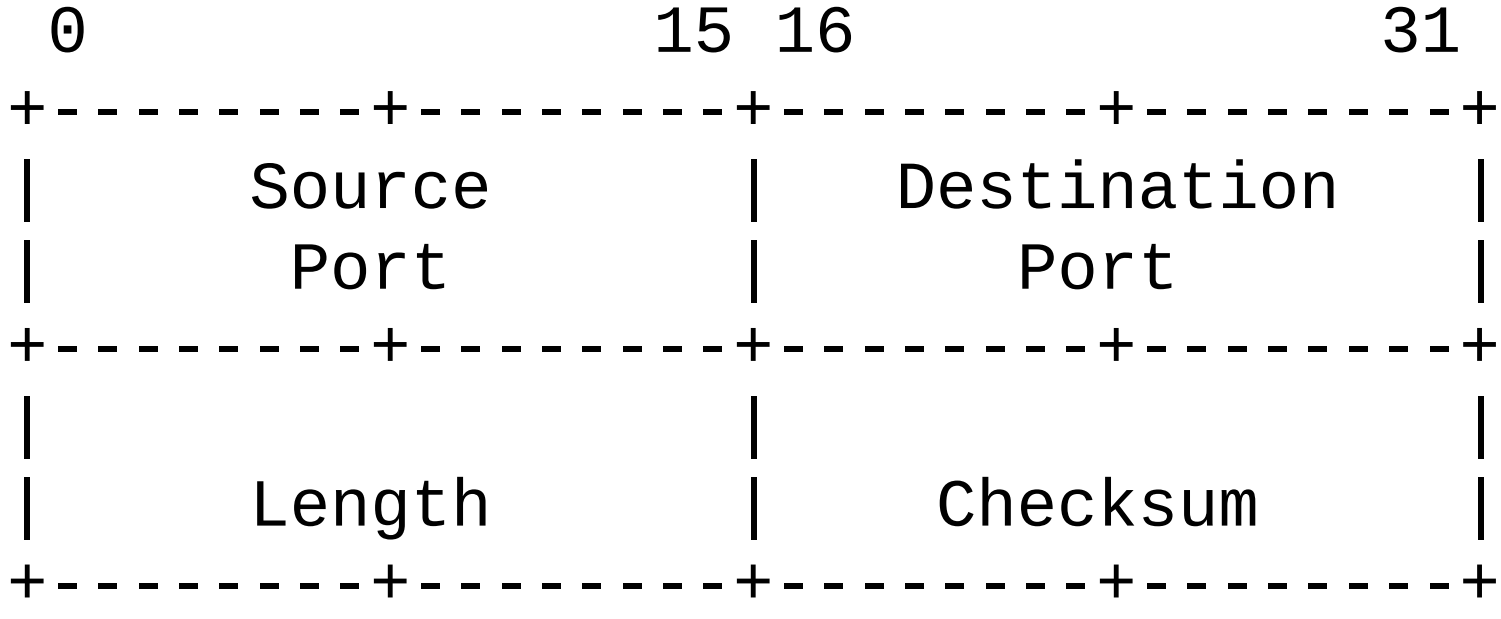

0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Version
IHL
Type of Service
Total Length
Identification
Flags
Fragment Offset
Time to Live
Protocol
Header Checksum
Source Address
Destination Address
Options
Padding

IP defragmentation

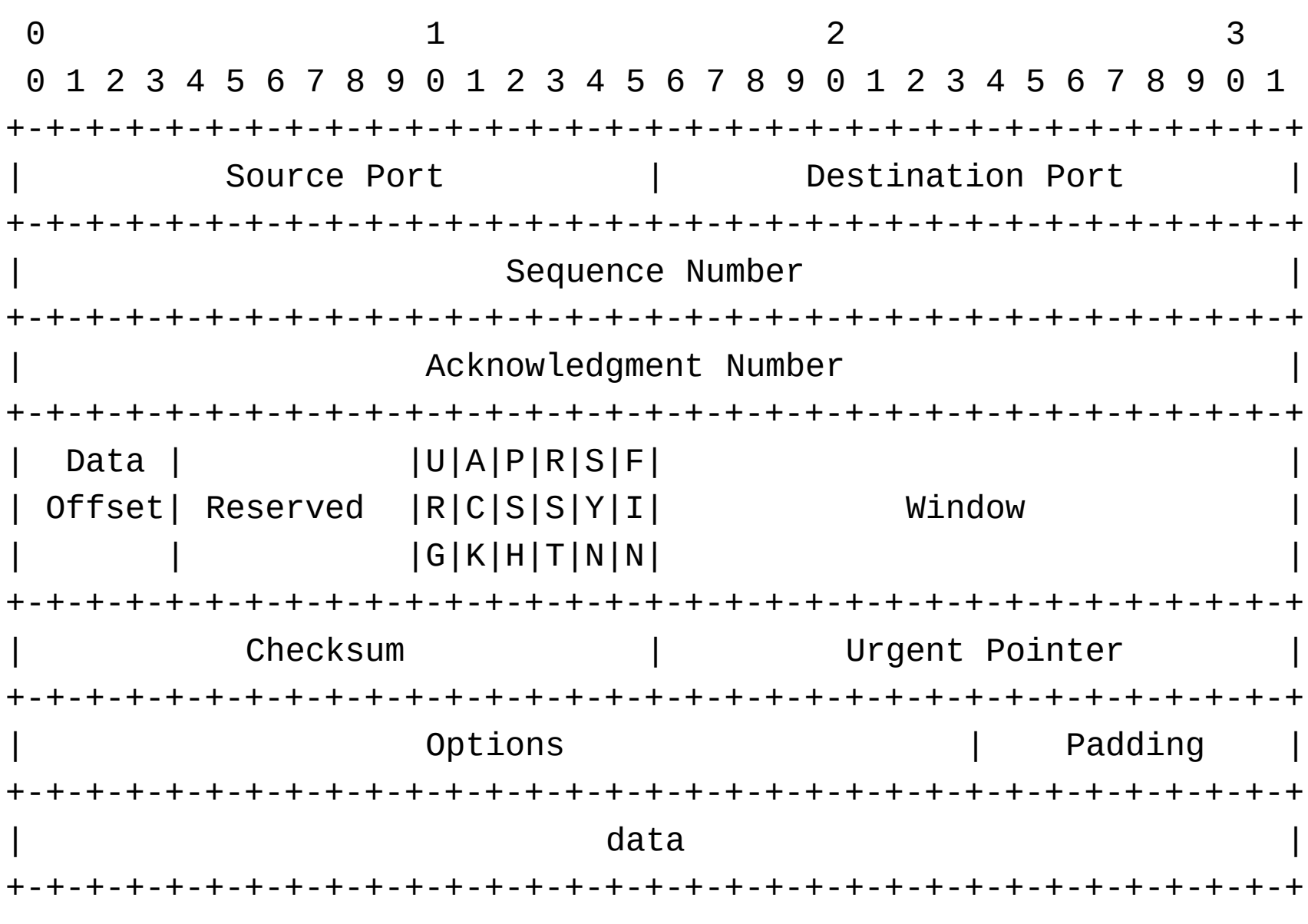
defrag criteria

- first packet’s offset is 0
- last packet’s ’last’ flag is true
- packet is continous

Why i like UDP



the TCP mess


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Data Offset
Reserved
U R G
A C K
P S H
R S T
S Y N
F I N
Window
Checksum
Urgent Pointer
Options
Padding
data

reconstructing streams

a stream must

- begin with a 3 way handshake
- every received data packet must be ACK'd
- sequence must be continous
- must reset on RST
- must close direction on FIN (which must be ACK'd)

implementing this

one method of implementation

- open a new stream on SYN
- have a state, to treat handshakes and FINs
- have 2 un-ack’d queues
- when data is ack’d, stick the data in a buffer
- on RST destroy

to summerize

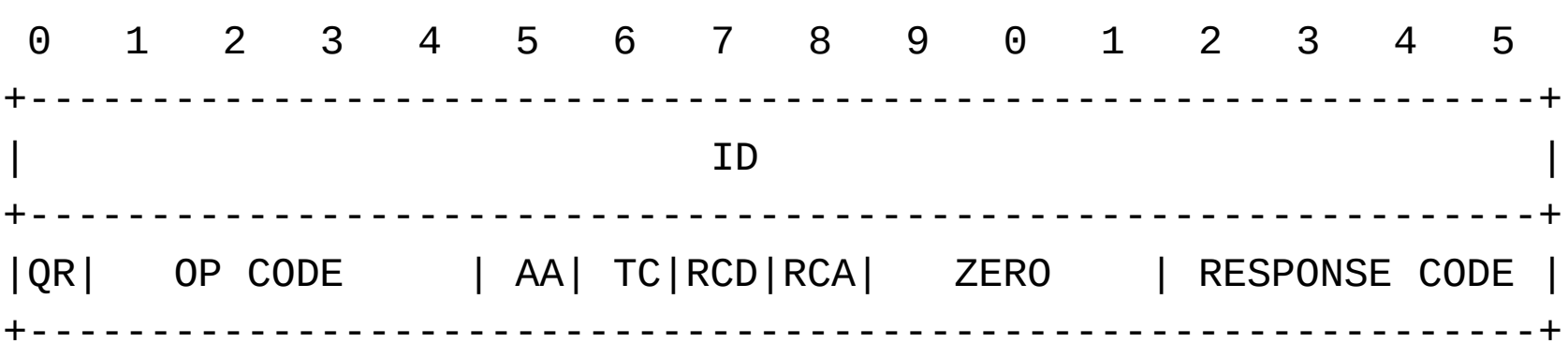
so when we get a packet, we:

- get packet
- defrag packet(s)
- if UDP, done!
- if TCP, toss in stream
- check stream for new data

the messy DNS system

DNS is a messy system, as you will see

DNS packet format



Detour: Messing with the bindings

- Problem: findalldevs isn’t bound
- Solution: Bind it (cue the patch)
- Patched in Ubuntu Dapper, but not upstream :(