Learning From Bugs

Henrik Warne, SAST Stockholm, Q2 2019

About me

- Software developer at TriOptima in Stockholm
- Previously: Symsoft, Tilgin, Ericsson
- Programming for more than 25 years
- Currently: Python
- Previously: Java, C++, PLEX
- @henrikwarne

1. Tracking interesting bugs

- What bugs are interesting?
- Why – helps me learn, can review later
- How – entries in bugs.txt (a plain text file)

Entry template

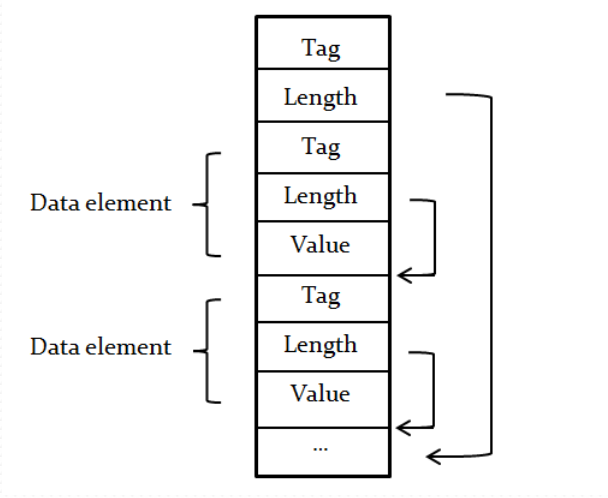
-------template begin----------
Date:
Symptom:
Cause:
How found:
Fix:
Fixed in file(s):
Caused by me:
Time taken to resolve bug:
Lessons:
---------- bug separator ----------

-------template end----------

Example bug (1)

Tag-length-value (TLV) messages.

Use length to skip unwanted data elements.



Example bug (2):

Date:

2004-08-17

Symptom:

Infinite loop when decoding Q.931 message

Cause:

When an unknown element id is found in a Q.931 message, we try to skip it by reading the length, and advancing the pos pointer that many bytes.

However, in this case the length was zero, causing us to try to skip the same element id over and over.

Example bug (3):

How found:

This happened during parsing of a setup message taken from an Ethereal trace from Nortel. Their message was 1016 bytes long (it included a lot of fast start elements), but our MSG_MAX_LEN was 1000. Normally we then receive a truncated message from common/Communication.cxx, but now, when fed directly in to be parsed, memory past the end of the array was accessed, and it happened to be zero, exposing this problem.

To find it, I just added a few print outs in the q931 parsing code. But it was lucky that the data happened to be zero.

Example bug (4):

Fix:

If the length given is zero, set it to one. This way we always move forward.

Fixed in file(s):

callh/q931_msg.cxx

Time taken to resolve bug:

1 hour

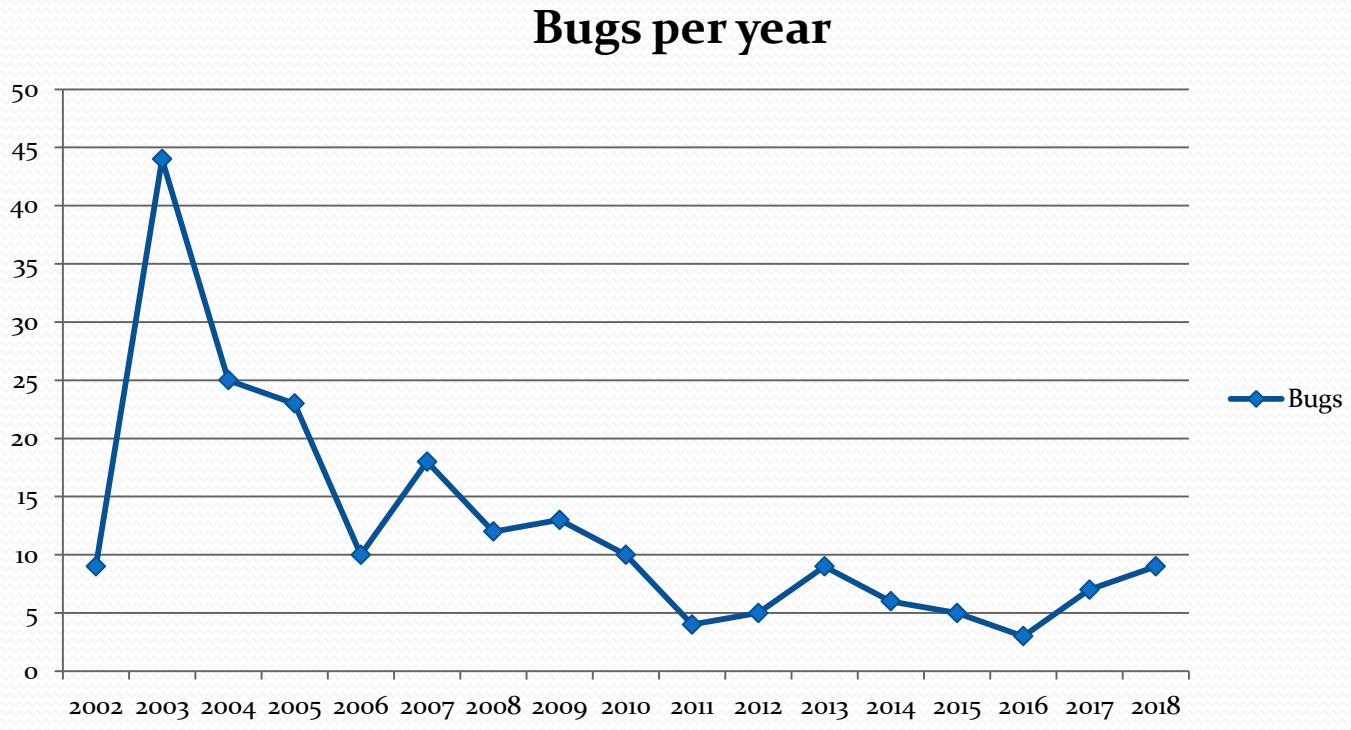
Lessons:

Trusted the data received in an incoming message. It's not just giving huge numbers that can cause problems. Indicating a length of zero could be just as bad.

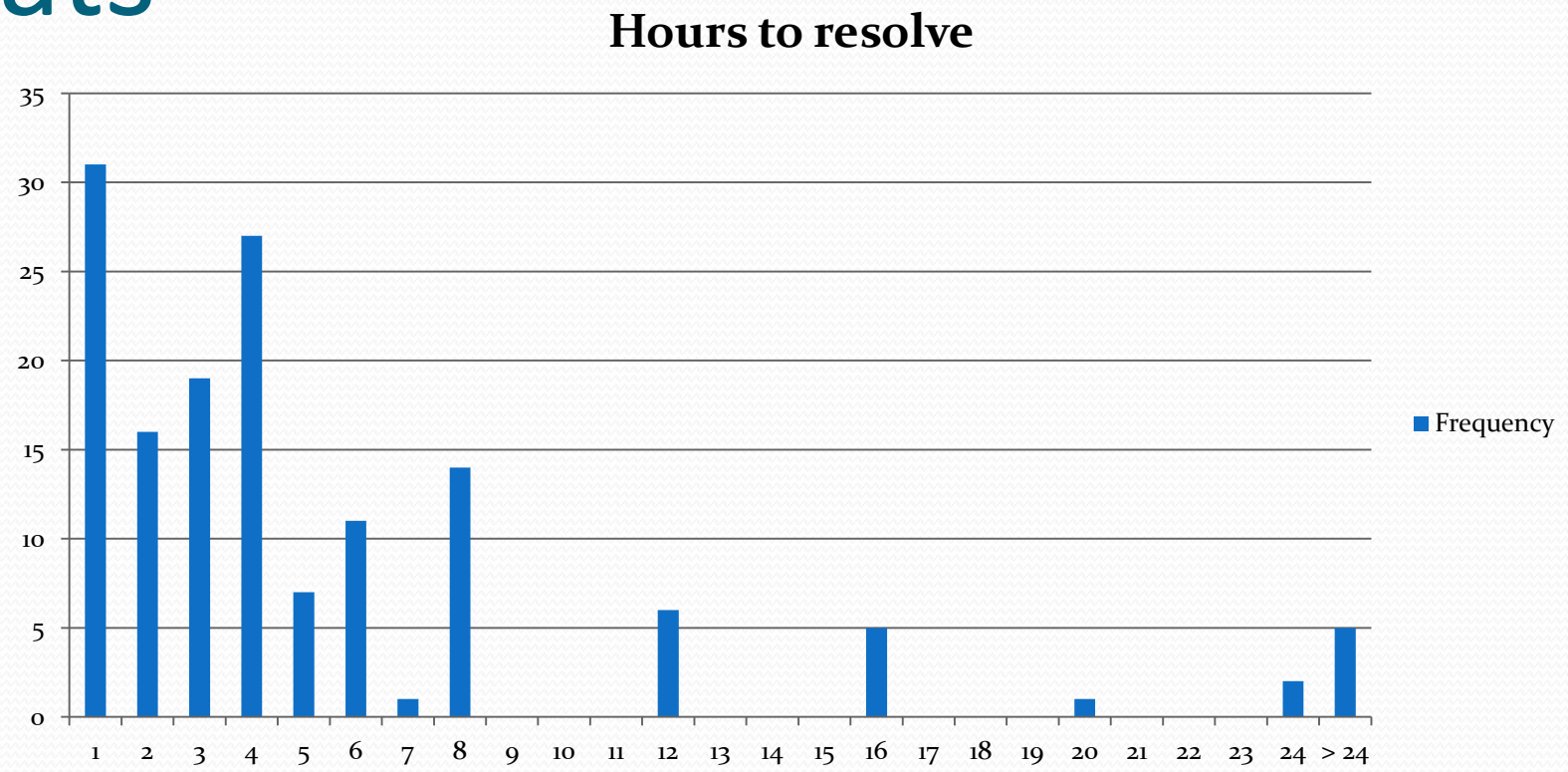
Tips

- Write it down as soon as possible (< 1 day after)
- Act of writing clarifies thinking – try it
- You decide what bugs are interesting

Stats


Bugs per year
50
45
40
35
30
25
20
15
10
5
0
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
Bugs

Stats


Hours to resolve
35
30
25
20
15
10
5
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
> 24
Frequency

2. Lessons (15 years, 200 bugs)

Coding

- **1. Event order**
  Arrive in a different order?
  Never received?
  Twice in a row?
  Not normally, but bugs in other parts can make it happen

- **2. Too early**
  Special case of event order
  Traffic before all config is done
  Put in a list – already marked as ”down”

Coding

- **3. Silent failures**
  Everybody knows – but still happens
  Check result of system calls
  Explicit error – then add ways to recover

- **4. If**
  Nested if:s
  if (a or b), if (x) else if (y), if (x, y, z) if (x, z, w)…
  Rewrite to simplify, unit test

Coding

- **5. Else**
  Usually when if, also need else
  Related: setting a flag – when should it be cleared

- **6. Logging**
  Not too much, not too little
  Visibilty needed when things don't work

Coding

- **7. Changing assumptions**
  Easy to change all code dependencies
  Hard to find all implicit assumptions
  When should this be detected?

Testing

- **8. Zero and Null**
  String – both null and length zero
  Nothing – no bytes sent over TCP connection
  Both in automatic checks, exploratory tests

- **9. Add and remove**
  New config profile – so create and add
  Try removing as well

Testing

- **10. Error handling**
  Hard to test – separate action from triggering it
  Flip a condition – if error_count > 0 to == 0

- **11. Random input (fuzzing)**
  Only applicable in some cases, but big payoff
  H.323 binary encoding of messages
  Generate randomized phone calls

Testing

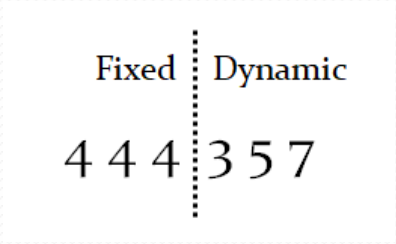
- **12. Check what shouldn't happen**
  Natural to check what should happen – but also reverse

- **13. Own tools**
  SIP protocol testing – customized (valid) replies
  Command line tool for API calls
  Start small, gradually add more

Testing

Testing wont find all bugs.

Routing numbers – first dynamic digit lost. Worked 100 times, then failed 900 times.



Debugging

- **14. Discuss**
  By far most common lesson
  Especially for hardest bugs
  Colleagues don't even have to know code in question

- **15. Pay close attention**
  Often made false assumptions
  Other exception thrown
  Never got to method I thought
  Different version of the SW running
  Faulty printout – print "a=%s" % b

Debugging

- **16. Most recent change**
  Only logging – but NullPointerException for message
  Could be a merge with old commits
  Continuous delivery helps – smaller deltas

- **17. Believe the user**
  ”Impossible – must have done something wrong”
  Used in unanticipated way

Debugging

- **18. Test the fix**
  Recreate the problem (if you can't, means something)
  Apply the fix
  Problem gone

Recap

- Easy to do – use a simple text file
- Makes you reflect on the bugs
- What lessons are learnt
- Optional: review periodically

Questions?