Don’t Feed the Bugzilla: Squash (Heisen)bugs Before Release

Klaas van Gend

The dynamic analysis experts

klaas@vectorfabrics.com
http://www.vectorfabrics.com
Quiz: Find the 5 issues in below pseudo-C code?

Thread 0

```c
int t1 = t2 = sum = 0;
spawn Thread 1;
spawn Thread 2;
while (!t1 && !t2)
  /* nothing */
print sum;
```

Thread 1

```c
repeat 100:
  sum++;
t1 = 1;
```

Thread 2

```c
repeat 100:
  sum++;
t2 = 1;
```
Woken up? The answers!

1. BUG: && should be ||

2. `sum++` translates to load-inc-store
   - not thread-safe and constitutes a data race.

3. Loads/stores of `t1` and `t2` can be optimized away by the compiler (to processor registers).
   - The `while` will hang and print nothing at all.
   - The `while` may be optimized away as well
   - `volatile` can address the hang but not many other issues.

4. A non-sequential memory consistency model of the processor can break the ‘barrier’ based on `while`.

5. Code likely to run slower than sequential version
   - amount of overhead versus little workload (200 increments).
Outline

• What’s a Heisenbug?
• How do you find (Heisen)bugs?
• When/Where do you find bugs?
• Summary

Throughout the presentation, we will show examples of bugs found in mature, professional, high-quality open source projects
What is a Heisenbug?
What’s a “Heisenbug”? 

In computer programming jargon, a **Heisenbug** is a software bug that seems to disappear or alter its behavior when one attempts to study it. The term is a pun on the name of Werner Heisenberg, the physicist who first asserted the **observer effect** of quantum mechanics, which states that **the act of observing a system inevitably alters its state.**
The Art of Bugfixing

How to Debug Heisenbugs

Don't look at the screen!!!!
## Heisenbugs: Typology

### General problems:
- Timing Issues
- Use of Stale Data
- Use of Random Data

Usually “reproducible”
Not “debuggable”

### Causes:
- Use of Uninitialized Data
- Buffer Overflows
- Use after Free
- Data Races
- Memory Ordering issues
- Debug-Release Differences
- TOCTOU
Bug categories

Deterministic
problem always appears at fixed conditions

Non-deterministic
problem appears "at random"

Debuggable
printf or gdb do not alter the bug's appearance

Non-debuggable
printf or gdb are not helpful

Infinite recursion
Wrong data
Hangs
Easy
Buffer overflows
Uninitialized variable use
Debug/release differences
Heisenbugs
Real-time issues:
* I/O skips
* Deadlines

Deadlocks
Interface
Unprotected accesses
Multi-threading bugs
Uninitialized variable use
Livells
TOCTOU
Data races
Navigation software

Features:
- Map display in 2D
- Route planning
- Route guidance
- Speech instructions

Made effort to get high quality code:
- Open source: peer reviewed
- Code quality confirmed by Coverity (static analysis)

http://sourceforge.net/projects/navit/
1) Read from uninitialized stack object

```c
void tracking_update(struct tracking *tr, struct vehicle *v, struct vehicleprofile *vehicleprofile, enum projection pro)
{
    ...
    struct coord cin;
    ...
    transform_distance_line_sq(&sd->c[i], &sd->c[i+1], &cin, &lpnt_tmp),
    ...
}
```

- Stack object is NOT set to zero
- `cin` passed as `ref`
- Memory read here
- Decision here
- Side effects
2) Use after deallocation

```c
binmap_search_new() , navit/map/binfile/binfile.c 2128-2140

... map_rec = map_rect_new_binfile(map, NULL);
town = map_rect_get_item_byid_binfile(map_rec, map->last_searched_town_id_hi,
    map->last_searched_town_id_lo);
...

map_rect_destroy_binfile(map_rec);

if (msp->boundaries)
    dbg(lvl_debug, "using map town boundaries\n");

if (!msp->boundaries && town)
    {
        binmap_get_estimated_boundaries(town, &msp->boundaries);
        if (msp->boundaries)
            dbg(lvl_debug, "using estimated boundaries\n");
    }
```
Dynamic Analysis is a good companion

We (Izm) filed a bug: [http://trac.navit-project.org/ticket/1316](http://trac.navit-project.org/ticket/1316) NAVIT developer (KaZeR) on IRC #navit; Aug 26 2015:

< KaZeR> Izm: thanks for that bug report!
< KaZeR> we are also using Coverity for this kind of analysis
< KaZeR> interesting.. I can't find this one in our Coverity's report
<lzm> probably because Coverity does static analysis on the source code, while Pareon Verify does dynamic analysis on a running program on the target platform (x86/arm/android)
< KaZeR> which makes it a really good companion

Bug was fixed day after reporting
How do you find (Heisen)bugs?
How to find Heisenbugs?

Heisenbugs are “non-debuggable”:
- Direct observation doesn’t work

Solution: use ‘non-invasive’ tools:

- **Static analysis:**
  - Compile time: parse code and understand flow
  - Examples: cppcheck, PClint, Coverity, Klockwork, CodeSonar, QAC

- **Dynamic analysis:**
  - During execution, analyze (instrumented) binary
  - Examples: Valgrind/Memcheck/Helgrind, TSAN/MSAN/ASAN, Vector Fabrics’ **Pareon Verify**

Static Analysis vs Dynamic Analysis

Source code

Compile → Executable → Execute → Output

Static analysis

Compile → Error report

Dynamic analysis

Execute → Error report

Error report
“JMDdecode” H.264 reference software

• H.264 - video coding standard
• Golden reference implementation maintained by Fraunhofer
• Mature open-source project
• 117K lines of C
• http://iphome.hhi.de/suehring/tml
H.264 reference software

```c
char INIT_FLD_MAP_I[1][8][15][2];
...
IBIARI_CTX INIT2 (NUM_BLOCK_TYPES, NUM_MAP_CTX,
|   |   |   |   |
| tc>map_contexts[1], INIT_FLD_MAP, model_number, qp);
...
int pstate = ((ini[0] * qp) >> 4) + ini[1];
```

value 22, in macro mapped to 2nd index

- Multi-dimensional array
  - accessed inside its boundaries
  - with a wrong index
  - getting wrong data via a pointer
  - from a valid data element

- Test case succeeds!
Pareon Verify error message

[M0193] Static-buffer overflow(s) detected:

the read in

function biari_init_context at biaridecod.c:299
called from function init_contexts at context.ini.c:90
called from function decode_one_frame_image.c:943
called from function DecodeOneFrame at ldecod.c:1254
called from function main at decoder_test.c:245

performed 210 access of size 1 between the offsets of 240 and 658 bytes of
the object of size 240 allocated as `INIT_FLD_MAP_I' at ctx_tables.h:879

where array index 21 is outside of array `INIT_FLD_MAP_I[][0..7][][]'

in

function init_contexts at context ini.c:90
called from function decode_one_frame at image.c:943
called from function DecodeOneFrame at ldecod.c:1254
called from function main at decoder_test.c:245

• Exact specification of the faulty index in the array
• Bug fixed two days after the submission
• https://ipbt.hhi.fraunhofer.de/mantis/view.php?id=348
Where/When to find bugs?
Where do you find bugs?

In a bugtracker, because users found it…

- “ABC doesn’t work”
- “I click on the OK button, but it’s not OK”

- Of course, users that know how to write good bug reports DO exist!

But do you want your users to find *your* bugs?
Continuous Integration

Source code

Compile

Executable

Execute

Output

Static analysis

Error report

Dynamic analysis

Error report
Continuous Integration

Continuous Integration tools, like Jenkins:

After each check-in or every night:

- Checkout,
- Compile application + tests
- Run all tests
- Scream if tests break (i.e. tests return “fail”)
Cost of a bug (commercial environment)

The cost to find and fix a defect

System test
Integration test
Unit testing
Code inspection

Design  Coding  QA  Production
Find Heisenbugs before they appear

Bugs found when running unit/integration tests with dynamic analysis:
• The night after commit
• Developer still has the code fresh in mind
• Instead of e.g. 4 months later

Current state of open source dynamic analysis tools for use in regression runs:
• **Valgrind/Memcheck/Helgrind:**
  – huge memory overheads on test PC,
  – output interspersed with test output, need to adjust test code
• **Google (T/A/M)SAN:**
  – stop at first error found – no indication how many errors would be found in total run
• If many test binaries need mechanism to merge reports
picoTCP is the answer for a size, speed and feature conscious open source TCP/IP stack for embedded devices.

https://github.com/tass-belgium/picotcp
High Quality of PicoTCP

- Deploy Continuous Integration
- Have an excellent set of unit tests and integration tests
- Have a (manual) set of tests that cover the applicable RFCs really well
- Use code coverage measurements to improve and add new tests
- Use static analysis tools and fix all issues

- Yet still real issues left in the code
TCP/IP stack on GitHub

Patch:
- Global variable moved to stack
- Pointer to stack variable passed to a callback function
- Pointer dereferenced after stack deallocation
TCP/IP bug fixes on GitHub

**SNTP: Fixed short allocation for server string**

When the sntp cookie is created, the server string was being allocated using strlen(), which would overflow when a strcpy is used from the same source by putting the string terminators out of the allocated object bounds.

Bug discovered by Pareon Verify.

---

**IPv4: Check packet len before processing**

When a packet is received, the length in the header must be checked against the actual IP buffer length. If the header length has been altered, or it's set to a bigger value on purpose by an attacker, the CRC function may violate the heap memory boundaries.

Discovered using Pareon Verify.

---

`tass-belgium / picotcp`
“Visualization ToolKit”

a.k.a. “VTK”
- Mature OSS project
- 1.5M lines of C++
- 800k lines of C
- http://www.vtk.org
Visualization ToolKit

```c
memcpy (cword, "abcdefgh", 8);
Swap2BERange(cword, 8);
...
void Swap2BERange(T* first, size_t num)
{
    // Swap one value at a time.
    T* last = first + num;
    for(T* p=first; p != last; ++p) {
        Swap(reinterpret_cast<char*>(p));
    }
}
```

- Pointers and casts, C++ templates and classes
- But the code looks OK and...
- Test succeeds

T is not char!
[M0203] Read(s) from uninitialized stack object detected:
the read in
  function vtkByteSwapper<2ul>::Swap at vtkByteSwap.cxx:43
called from function vtkByteSwapRange<short> at vtkByteSwap.cxx:75
called from function vtkByteSwapBERange<short> at vtkByteSwap.cxx:193
called from function vtkByteSwap::SwapBERange at vtkByteSwap.cxx:240
called from function vtkByteSwap::Swap2BERange at vtkByteSwap.cxx:298
called from function TestByteSwap at otherByteSwap.cxx:57
called from function otherByteSwap at otherByteSwap.cxx:160
called from function main at vtkCommonCoreCxxTests.cxx:372
performed 1 access of size 1 at an offset of 8 bytes from the start of
the stack object of size 1024 allocated as `cword' in
  function TestByteSwap at otherByteSwap.cxx:32
called from function otherByteSwap at otherByteSwap.cxx:160
called from function main at vtkCommonCoreCxxTests.cxx:372

- Easy patch based on the clear error message
- Bug reported in 6.1.0 and fixed in 6.2.0
Summary

Don’t Feed the Bugzilla: Squash (Heisen)bugs Before Release
1. Code is never bug-free, don’t let your users fill up your bug tracker

2. Static errors are relatively easy to catch, many professional open source projects work on quality; dynamic errors slip through or are found late in release

3. Dynamic Analysis is a “really good companion” to static analysis, especially in Continuous Integration

4. Serious effort needed to beef up existing open source tools for deployment in CI

   Workable commercial tools exist, from Vector Fabrics, with references from open source
Questions?

1. Code is never bug-free, don’t let your users fill up your bug tracker
2. Static errors are relatively easy to catch, many professional open source projects work on quality; dynamic errors slip through or are found late in release
3. Dynamic Analysis is a “really good companion” to static analysis, especially in Continuous Integration
4. Serious effort needed to beef up existing open source tools for deployment in CI
    Workable commercial tools exist, from Vector Fabrics, with references from open source

klaas@vectorfabrics.com