Everything about bugs

Wei Le

August 24, 2023

Outline

- Importance
- Examples
- Terminologies related to bugs
- Finding bugs

Why should we care about bugs?

- software bugs cost 1.7 $trillion in financial loss in 2017 [1]
- impact billions of people
- consequential bugs
- bugs are unavoidable: e.g., 391 commits of bugs, 287 commits of other stuffs in one of our studies.

[1] https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses-in-2017/

Top software failure recently [2]

- February 2020: Heathrow disruption: cannot check in e-tickets
- Nest thermostat leaves users in the cold: battery drain
- Starbucks's software bugs: 60% stores in US and Canada have to be closed and give coffee away for free as they cannot process payment transactions
- Co-op charges customers twice
- March 2015: F-35 detects targets incorrectly
- 2014, 2015 Toyota recall its cars: bugs on engine control unit, overheat transistors, can cause sudden stops while driving
- Emergery calls go offline for 6 hours

[2] https://www.computerworld.com/article/3412197/top-software-failures-in-recent-history.html

Top software failure in history

- Therac-25 (radiation therapy machine) had $\geq$6 incidents between 1985-1987 and gave patients radiation doses that were hundreds of times greater than normal, resulting in death (in 3 cases) or serious injury
- written in assembly language
- had both design problems and coding problems including race conditions, arithmetic overflow
- more on wiki page "Therac-25"
- loss of rockets and satellites: NASA Mariner 1 destruction (1962), Airane 5 flight 501 destroyed (1996), Mars climate orbitor

Examples

find.66c536bb bug [3]

find -mtime allows to find file according to their age. However, the buggy version interprets -mtime -n and +n in the same way. But it should interpret find -mtime -n as finding files that are *strictly less* than n days old and find -mtime +n as finding files that are *strictly more* than n days old.

```
$ mkdir tmp
$ touch -t 202208231600 tmp/a //today
$ touch -t 202208221600 tmp/b //yesterday
$ touch -t 202208211600 tmp/c //2 days ago
```

If we run the following, we would expect this output

```
$ ./find tmp -mtime -2
tmp
tmp/a
tmp/b
```

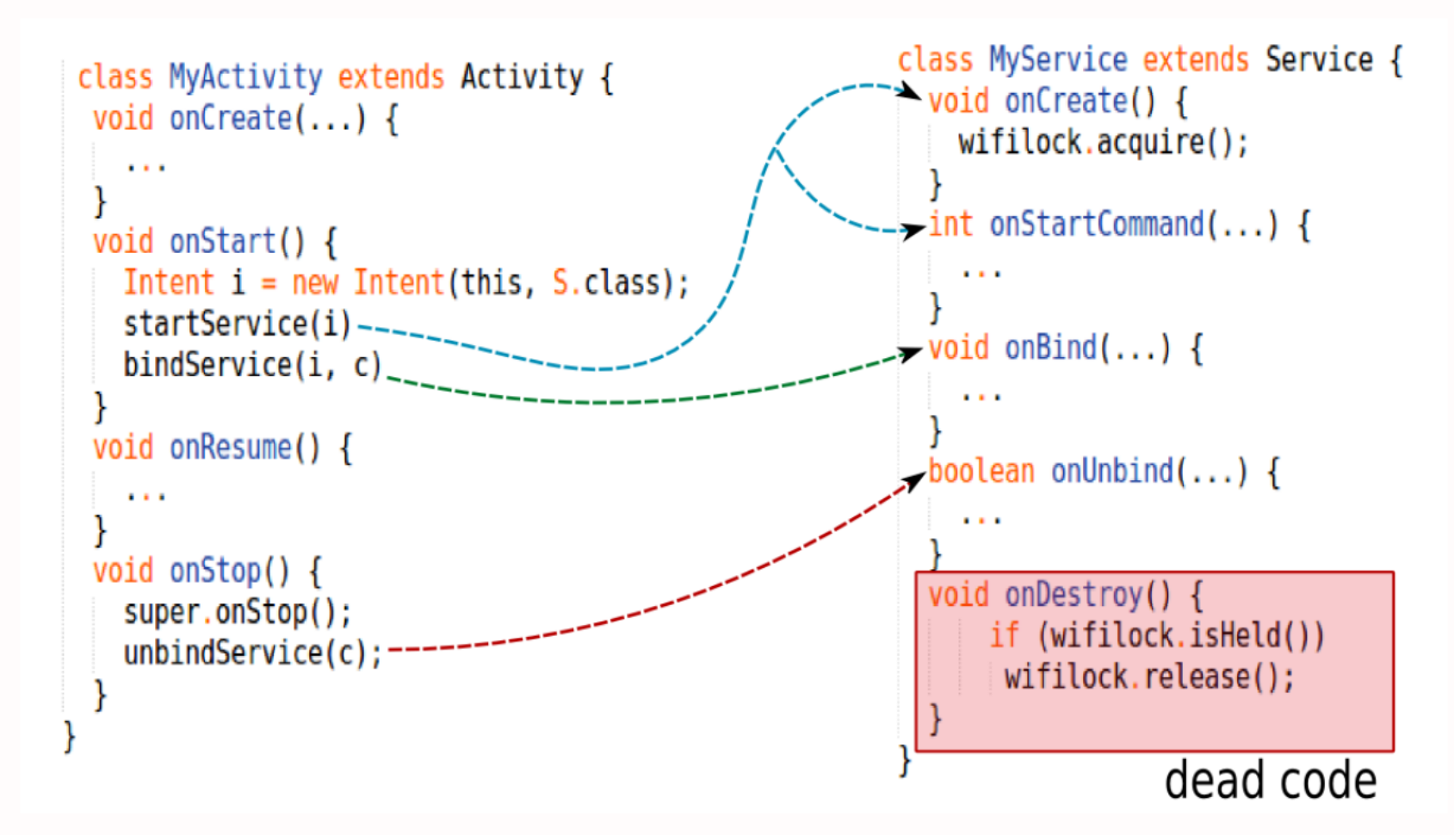
However, I actually get this output:

```
$ ./find tmp -mtime -2
tmp/c
```

Results are the same if I replace -n with +n, or just n.

[3] https://dbgbench.github.io/find.66c536bb.report.txt

Android bug: causing battery drain



Android bug: causing battery drain

```
1    class HostListActivity extends Activity {
2     public void onStart() {
3       this.startService(new Intent(this,
4         TrackingRecordingService.class));
5       this.bindService(new Intent(this,
6         TrackingRecordingService.class)),...);
7     }
8     public void onStop() {
9       super.onStop();
10      this.unbindService(connection);
11    }
12  }
13  class TrackingRecordingService extends Service {
14    public void onCreate() {
15      wifilock.acquire();
16    }
17    public boolean onUnbind(Intent intent) {
18  +   if (bridges.size() == 0) this.stopSelf();//patch
19  +   return true;
20    }
21    public void onDestroy() {
22      if (wifilock != null && wifilock.isHeld())
23        wifilock.release();
24    }
25  }
```

Terminologies

- *bug*: mistakes in software (code, configuration, makefile)
- *vulnerability*: is it exploitable? what is the threat model?
- *fault*: violation of *program property* – conditions hold for all program paths, e.g., *assertion*
- *failure*: dynamic symptoms - crash, incorrect results ... the crash stacks and memory states at the crashes can be captured and reported for postmortem analysis
- *root cause*: what is the mistake and how it is propagated and lead faults and failures
- *failure-inducing input*: inputs that can trigger the bug
- *reproduce steps and environment*: how to reproduce the bugs: in addition to inputs, we sometimes also need to know the libraries and system setups
- *patch, program fix*: the modification of code that ensures correct executions

Terminlogies: Types of bugs

Coding errors

1. buffer overflow, integer overflow, null-pointer dereference, double free, memory leak
2. deadlock, race conditions – concurrent bugs
3. lock/unlock mismatch, file open/close mismatch – resource leaks, typestate violations, source-sink problems
4. program specific, functionality issues

Web programming vulnerabilities

1. Cross site scripting
2. Path traversal

Top 25 CWE bugs

Bug in special types of software: *active research areas*

1. finding bugs in compilers, virtual machine software
2. finding bugs in machine learning code
3. finding bugs in UAVs

Finding Bugs

Bugs and software development lifecycle

The early we find bugs, the cheaper to diagnose and fix bugs. According to the search at IBM [4]:

[4] https://www.celerity.com/the-true-cost-of-a-software-bug

Current approaches for finding bugs

There are both static (don't need to run the program) and dynamic approaches (need to run the program) to find bugs

- *Static analysis* aims to predict such conditions by analyzing the source code.
- *Testing* aims to find such conditions by exercising representative inputs.
- *Dynamic analysis* collects the run time information to determine if a bug has been triggered. Dynamic tools can combine with static analysis tools and testing.
- *Model checking* builds models of software and check it against given properties.
- *Machine learning* finds bugs new

Current approaches for finding bugs

- *Code review*, *code inspection* finds bugs manually to confirm static warnings, to diagnose a failure. The big company like Microsoft prepare a very user-friendly GUI for code inspection.
- A typical process: static analysis/code inspection on the desktop $\rightarrow$ git commit and push to the clouds $\rightarrow$ unit testing / whole program static analysis /team code inspection/ integration testing $\rightarrow$ fuzzing
- *Continous integration*: nightly analyzing and testing changes, providing timely feedback to the developers

Automatic bug finding tools

Software companies such as Google, Microsoft, Facebook have deployed automatic tools; there are also bug finding tool companies:

- Static analysis tools: GammarTech (CodeSonar), Coverity, Fortify, PolySpace, ESP, ESPx, Infer (open source), Findbugs
- Testing tools: AFL (fuzzing tools), SAGE (dynamic symbolic execution)
- Dynamic analysis tools: Purify, Valgrind, AppVerifier
- Model checking tool: SLAM (finding bugs in drivers that can cause blue screens)
- AI models: e.g., CodeBERT, LineVul (low precision and recall)
- Implementing an algorithm using a week, but building a tool that can handle real-world software will need to handle many engineering challenges

Automatic bug finding tools

Name	Language	Type of Tool	Note
Findbugs	Java	Static analysis	open source, UMD/Google
American Fuzzy Lop	C/C++	Fuzzer	open source
Prefix, Prefast	C/C++	Static analysis	Microsoft
ESP	C/C++	Static analysis	Microsoft
KLEE	C	Static + Dynamic	open source
Infer	Java, C/C++, Objective C	Static analysis	open source, facebook
CodeSonar	C/C++	Static analysis	UW/GrammarTech
Coverity	C, ..	Static analysis	Standford/Synopsys
Valgrind	C/C++	Dynamic analysis	open source
Atlas	C/C++/Java	Static analysis	Iowa State/EnSoft

Behind the scene: the key ideas

- finding bugs: a software engineering problem
- does the program contain an "erroneous/undesired" state?: a program analysis problem
- problem reduction: we need to define what is the "erroneous/undesired", i.e., we need to know the "specification of a bug" in order to detect bugs
  - Statically: a program state is the values of a set of variables at a program point; the set of incorrect values form an erroneous/undesired state. Static analysis tools use the conditions defined in Common Weakness Enumeration (CWE)
  - Dynamically: crash, hang, incorrect output

Behind the scene: the key ideas

- Challenge: software consists of a very large state space, we cannot enumerate all the states
- Static analysis: abstraction - reasoning about a group of executions, values
- Testing and dynamic analysis: sampling
- Model checking: prove the correctness of the software with regard to a given property, sometimes defined as *automata*, *precondition* and *postcondition*.
- AI: patterns in the buggy code