Everything about bugs

Wei Le

January 15, 2020

Outline

- Why we should care about bugs
- How to analyze bugs
- Tools for finding bugs
- Relevant program analysis concepts and algorithms (program analysis is the "mathematics" of software study)

Why we should care about bugs?

- software bugs cost 1.7 $trillion in financial loss in 2017
- $312 billion annually
- impact 1/3 household
- very consequential bugs
- bugs are unavoidable: e.g., 391 commits of bugs, 287 commits of other stuffs
- inconvenience, loss of money and productivity: e.g., faulty software recalls cars, cannot print the paper, google news app drains your data

Consequential bugs

- 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"

Consequential bugs

Loss of rockets and satellites:

Year	Events	PL	Root cause
1962	NASA Mariner 1 destruction	Fortran	Coding incorrect formula
1996	Ariane 5 Flight 501 destroyed		Arithmetic overflow
1999	Mars Polar Lander destroyed		
2000	Zenit 3SL launch failed		
2005	CryoSat-1 satellite lost		
	Mars Climate Orbiter		software on the ground generating commands in pound-force (lbf), while the orbiter expected newtons (N)

more consequential bugs are listed at
https://en.wikipedia.org/wiki/List_of_software_bugs

Impossible Bugs

Eitan Adler

1. MRI disabled every iOS device in facility
2. We can't send mail more than 500 miles
3. OpenOffice.org can't print on Tuesday (see comment 28)
4. I can't log in when I stand up. (and another similar story)
5. A story about "magic"
6. Print this file, your printer will jam
7. gcj crashes in April and December, but only if you speak German in Austria
8. Processor 5 doesn't work if you're standing too close
9. A car that is allergic to vanilla ice cream

Bugs and software lifetime

The early we find bugs, the cheaper to diagnose and fix bugs:

Analyzing software bugs: terminologies

- *bug*: mistakes in code (*vulnerability*: a bug that can be exploited)
- *fault*: violation of *program property* – facts hold for all program paths, e.g., *assertion, typestate*
- *failure*: dynamic symptoms - crash, incorrect results ... the crash stacks and memory states at the crashes sometimes ar reported
- *root cause*: explain what is the bug and how the error state introduced by a bug is propagated to lead to fault and failure, what types of bugs cause failure
- *test input* that can trigger the bug (input, or a sequence of events for GUI)
- *reproduce steps*: how to reproduce the bugs: in addition to test inputs, we also need to know which versions of libraries and environment setups
- *patch, program fix*: the modification of code that ensures correct executions
- *fault signatures*: see later slides

Fault signatures

Fault signature: a set of statements along a path that lead to the fault

Intuitively, we highlight statements only related to the bugs; people create benchmarks like buffer overflow benchmarks by MIT lab

See Examples/faultsignature.ppt

Types of bugs

Coding errors:

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

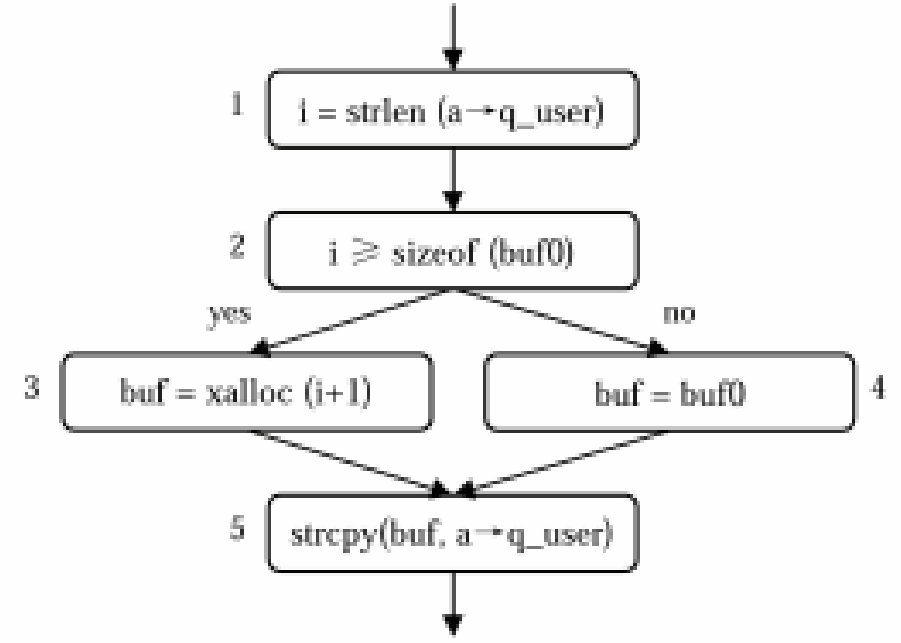
Bug in special types of software: *active research areas*

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

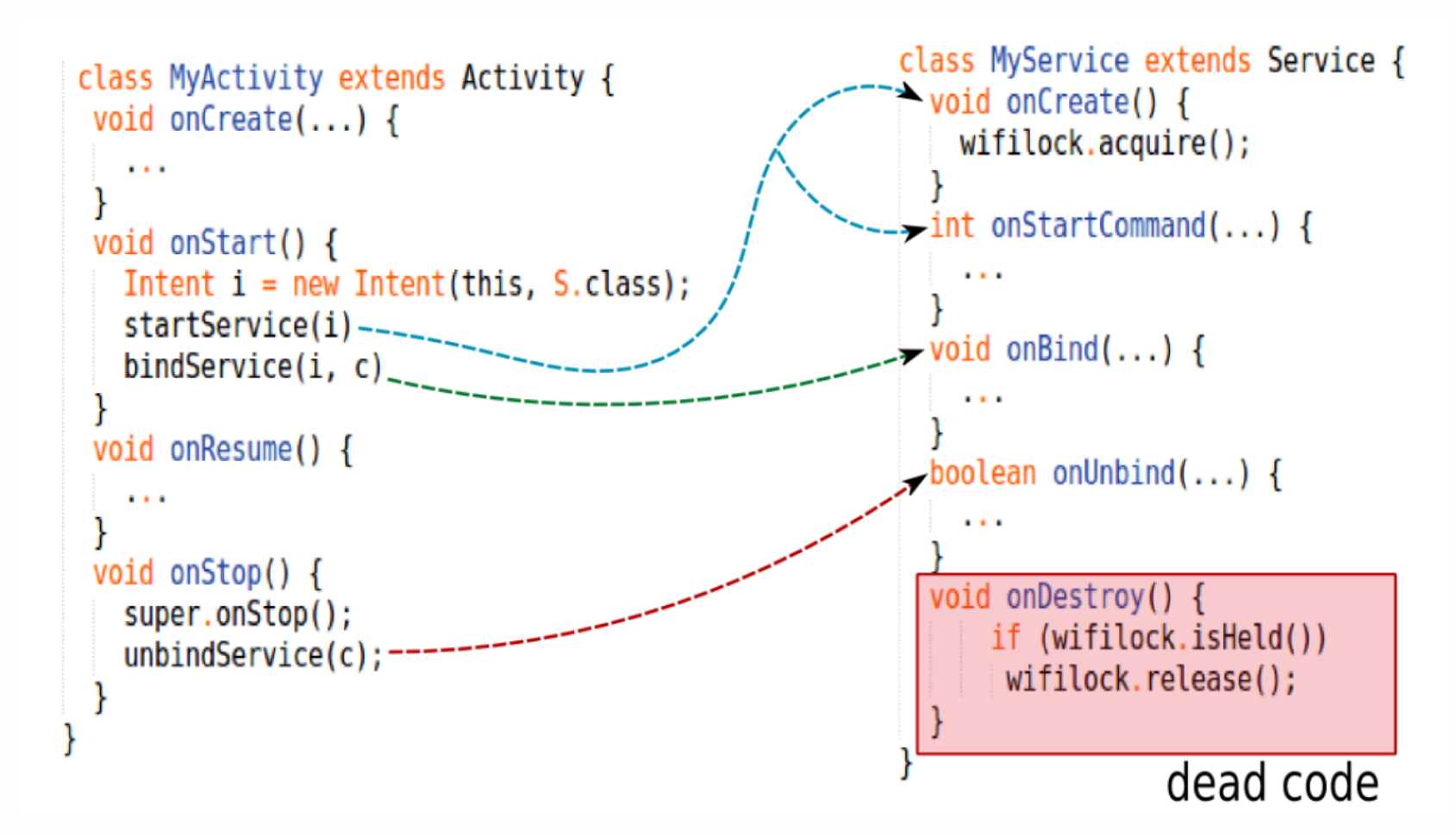
Analysis of bugs: Buffer Overflow

buffer overflow examples – is there a bug in the code?
why buffer overflows are bad? (corrupt stack and heap)

Analysis of bugs: Buffer Overflow



Analysis of bugs: Android wakelock bug causes battery drain



Analysis of bugs: Android wakelock bug causes 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  }
```

Where can you find information about real-world bugs

Software articrafts related to bugs:

- bug reports for open source software
  1. bugzilla: linux - https://bugzilla.kernel.org/query.cgi
  2. github issue trackers: numpy issue trackers - https://github.com/numpy/numpy/issues
  3. google issue trackers: Android framework - https://issuetracker.google.com/issues?q=componentid:192705%2B, https://source.android.com/setup/contribute/report-bugs#bug-queues
- CVE: https://www.cvedetails.com/product/3264/Mozilla-Firefox.html?vendor_id=452
- crash reports: https://crash-stats.mozilla.com/
- patches and pull requests on github
- bug benchmarks: e.g., bugbench by Shan et al.

Finding bugs in software and its relation with program analysis

Problem reduction: "finding bugs" (software engineering problem) is to ask "does the program potentially contain an "erroneous/undesired" state? e.g., crash, hang, incorrect output?" (program analysis problem):

- *program state*: a set of values of variables at a program point
- *program property*: the conditions that are true regarding these values, if there are any executions that potentially violate the program property, the fault exists in the code

Challenge: for general programs, it is infeasible to execute every input to find the erroneous conditions.

Common Weakness Enumeration (CWE)

cwe
119: Buffer Overflow
399: Resource leak
253: incorrect check of function return value

Current approaches for finding bugs

1. *static analysis* and *dynamic analysis* can both find bugs, software companies such as Google, Microsoft, Facebook has deployed automatic tools
   - Static analysis aims to predict such conditions by analyzing the source code. The key idea behind static analysis is abstraction.
   - 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.
2. *code review*, *code inspection* finds bugs manually to confirm static warnings, to diagnose a failure

Bug finding tools in use

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

Bug finding tools in research prototype stage

Name	Language	Type of Tool	Note
Saturn	C	static analysis	paths, lock/unlock
Marple	C/C++	static analysis	paths, buffer overflow, integer overflow ..
...	...	...	...

Further reading

1. Cost of software bugs
2. Summoning Demons: The Pursuit of Exploitable Bugs in Machine Learning
3. Exploiting a Buffer Overflow using Metasploit Framework
4. Finding memory leaks in C/C++
5. A few Billion Lines of code Later using static Analysis to find Bugs in the Real World

Catchup after homework

- *fault condition*: assertions you'll write to avoid the fault
  1. size(buf)<len(str): size computes the buffer size in terms of bytes, len counts the character in a string until '0' is encountered
  2. ptr == NULL
- *fault signatures*: see example fault signatures
  1. include the root causes
  2. remove not relevant code* (paths + cause)
  3. executable
- Bug location: where is the mistake in the code? sometimes can be the same as the patch, e.g., NULL-ptr checker. It is not always clear.
- Distances: a dynamic concept
  1. report N/A if it is located in different files
  2. count the instructions between bug and fault, fault and failure along the shortest paths