Using Symbolic Execution to Find Bugs in Code

Will O'Leary and Shadman Abedin
Mentor: Dr. Charles Wright, Portland State University

Motivation

- We rely on programs to keep of our critical information safe.
- A defect making a program crash, or **crash bug**, can create a security vulnerability.
- Modern programs are large and complicated, increasing the likelihood of crash bugs.
- Using a buggy program is frustrating, as well as unsafe.
- Thoroughly testing programs for crash bugs before release would be very beneficial.

Background

•A program is a list of instructions.

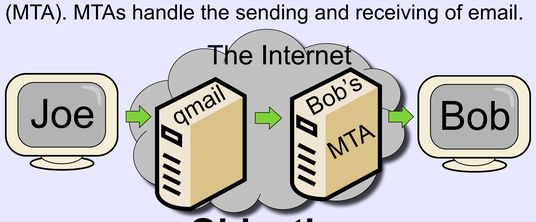
```
divide(a, b, do):
   result = 0
   if do is True:
      result = a ÷ b
   return result
```

divide(6, 3, True) → 2
divide(6, 3, False) → 0.

• A crash bug is an error in a programs code that causes it to stop unexpectedly.

divide(6, 0, True) → ERROR: Cannot divide by 0!

- The most cutting edge bug testing technique is **symbolic execution.**
  - Programs are run in a simulated environment with input that represents all possible inputs.
  - Symbolic execution is supposed to thoroughly test a program in a reasonable amount of time.
  - Symbolic execution has not yet been adopted by industry.
- We used Cloud9, a symbolic execution tool developed in 2011, to test the qmail program.
- Qmail is the second most popular mail transfer agent (MTA). MTAs handle the sending and receiving of email.



Objectives

- Thoroughly test qmail with Cloud9 and either find crash bugs or prove that qmail is crash bug-free.
- Determine if symbolic execution is developed enough for use in industry.

Abstract

We entrust software to protect much of our sensitive information. Therefore, software security is absolutely essential. However, modern programs are often so large and complicated that they almost always have crash bugs, or defects that make execution stop unexpectedly. Crash bugs can be exploited by malicious third parties in order to access personal data and to destroy vital systems. Furthermore, crash bugs are frustrating to the user. A method to test software for crash bugs before release would be very beneficial. We researched symbolic execution, a cutting edge technique to weed out crash bugs in large programs. In symbolic execution, programs are run in a simulated environment with inputs that represent all possible inputs. We tested qmail, a popular mail transfer agent, with Cloud9, a symbolic execution tool. Our goal was to thoroughly test qmail and either find crash bugs in the program or prove that the program was crash bug-free. More broadly, we wanted to evaluate whether or not symbolic execution is ready for use in industry. In the end, we found no bugs in qmail. Unfortunately, we only managed to test 17.65% of the program. Given all of our effort still translated to low coverage, we concluded that symbolic execution is still in its infancy and is not yet ready for use in industry.

Methods

(1) Source

```
int divide(int a, int b, bool do)
{
int result = 0;
if(do == true)
                    result = a/b;
return(result);
}
```

LLVM Compiler

(2) Bitcode

```
00010101
01010101
10101010
01010101
10101010
00100010
01001001
```

Cloud9

(3) Execution

-∞ < a < ∞
-∞ < b < ∞
do = true *OR* false

divide(a, b, do)

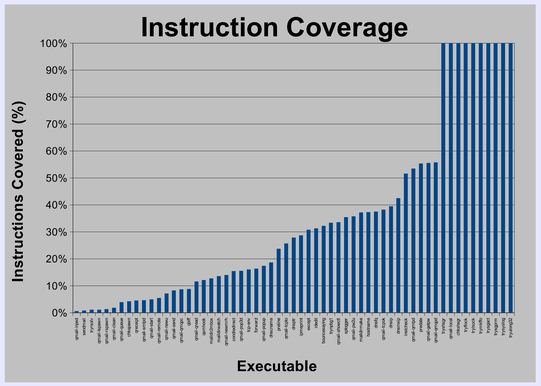
(4) Results

do = false
No error.

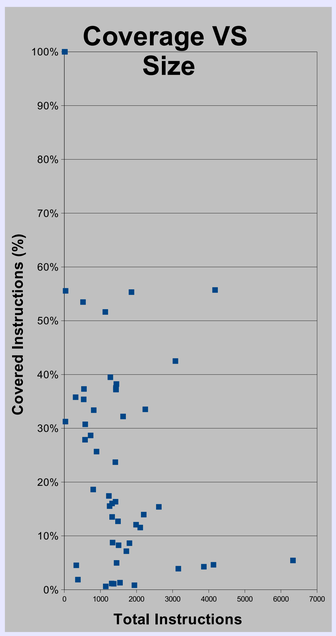
do = true & b ≠ 0
No error.

do = true & b = 0
ERROR: Cannot divide by 0!

Results



Overall Coverage: 17.65%
Bugs Found: 0



Conclusions

- We found no bugs in qmail.
- However, we only achieved 17.65% instruction coverage.
- The executables that achieved 100% coverage were actually part of qmail's existing test suite.
  - These programs were very small and simple, so they were easy for Cloud9 to test.
- The highest coverage on a "real" executable was 55.71% on qmail-local.
- There was no visible relationship between coverage and total number of instructions.
  - This suggests that Cloud9 did not simply need more time to test qmail.
- Coverage was low because of inadequate environment simulation and because we were not familiar with exactly how qmail works.
  - It was also low because our tests lacked network simulation.
- Symbolic execution is **not ready for industry.**
  - Current tools do not catch every common error.
  - Environment simulation is still not good enough.
  - Current tools are difficult to set up, maintain, and use.
  - Not all source code is easily converted into bitcode.
  - There is no support for programming languages other than C and C++.

Next Steps

- Gain deeper knowledge of the workings of qmail, and use that information to better simulate qmail's environment.
- Research a more complete way to simulating the environment.
  - For example, build a symbolic operating system or computer to totally simulate the environment.
  - Another possibility would be to use VirtualBox or a similar product to fool an existing Linux system to run a symbolic environment.
- Use design by contract programming.
  - Tell Cloud9 the expected inputs and outputs of every single function.
  - Then test individual functions instead of the whole qmail program.
  - This could speed up testing and increase coverage.

Acknowledgements

- Thanks to Dr. Charles Wright for giving us a research opportunity this summer and for helping us complete our project.
- Thanks to Dr. Karen Karavanic for providing us with lab space.
- Thanks to the Cloud9 team for responding to our questions.
- Thanks to Trimet.