Background

Software security is widely recognized to be an important problem with far reaching consequences. The cost of exploited security vulnerabilities in software systems has been widely documented, ranging from the loss of human lives to severe financial losses. This is the source of motivation for many researchers in software engineering, programming languages, and formal verification to develop methods for designing correct software systems, as well as detecting and fixing vulnerabilities in an existing system. This is also the starting point of the present PhD thesis.

Software security is by now a large multifaceted area of research. There are many important software artifacts that are around in the wild, some of which with serious security vulnerabilities (some, but not all of which, have been fixed). By the same token, redesigning new systems that are correct with the same functionalities is generally very expensive. So, this raises two interesting research problems for a software security researcher: (Q1) develop techniques for detecting vulnerabilities in existing systems, and (Q2) develop techniques for designing correct systems.

The present PhD thesis is ambitious in that Chen (the author) attempted to make substantial contributions towards advancing both of these questions, each of which is already a difficult research question in its own right. In particular, to address the first question Chen focussed on software testing since in this case one typically has to deal with low-level implementations (sometimes, in binary or assembly), for which verification techniques tend not to work that well in practice (owing to issues like scalability, and complex and subtle nuances of real-world languages). To address the second question Chen focussed on verification, especially type-checking based verification for information-flow security in Android apps.

This is a solid PhD thesis, three of whose chapters came from the author’s publications at CCS’18, CSF’18, and FSE’18, all of which are top conferences in computer security and software engineering.

Summary of Technical Contributions

The technical part of the thesis consists of four chapters.

[Chapter 3] To address Q1, the author contributed a new Directed Greybox Fuzzing (DGF) technique for low-level code (C/C++) implemented in a tool called Hawkeye. Greybox fuzzing is simply a term used for random testing methods that enhances blackbox fuzzing by performing lightweight program analysis in its preprocessing step. DGF is a class of greybox fuzzing techniques whose objectives are to reach certain locations (e.g. crash) in a program, instead of just path coverage. The state-of-the-art technique/tool for this is AFLGo (CCS’17), which reduces the testing problem to an optimization problem and uses simulated annealing to generate more seeds that are gradually closer to the target locations, and reducing the number of seeds that are further away. The thesis identifies valid problems with AFLGo, which prevent it from exposing some vulnerabilities in existing benchmarks. These include problems with the way distance to the target locations are calculated, which arose from their distance formula and the imprecision of the call graphs constructed by AFLGo. Chen addressed this issue by augmenting the distance function of AFLGo with information such as adjacent function distance, which could be calculated with the help of more precise static analysis methods (they use interprocedural value-flow analysis). Chen also observed that the mutation strategies of AFLGo are problematic in discovering certain bugs in that they can destroy seeds that are close to the target locations. Chen contributed an adaptive mutation strategy that varies across fuzzing states. He has shown experimentally that Hawkeye is superior to AFLGo and other methods in many cases (up to seven times faster than AFLGo).

[Chapter 4] Chen extended the technique from Chapter 3 for sequential programs to multithreaded programs resulting in the tool called Doublade. At the time of the thesis submission, the result was still under submission. The problem of testing concurrent programs is extremely difficult and is studied by many researchers in the community. DOUBLADE is the first greybox fuzzing method optimized for testing concurrent programs. The technical result involves making DFG aware of thread contexts. The experimental results of DOUBLADE are encouraging.

[Chapter 5] Hawkeye and Doublade were developed on top of a fuzzing framework, which Chen helped develop called FOT (published at FSE'18). Such a framework is extremely important since developing a new fuzzer from scratch requires a lot of development and implementation effort, which is time-consuming.

[Chapter 6] The issue addressed in this chapter concerns designing provably correct systems, and so formal methods are adopted. Chen developed an permission-dependent type system for information flow analysis. This is motivated by the issue of information leakage (i.e. non-interference) in Android apps, which is extremely important. The work builds on top of the previous work by Banerjee and Naumann. This work by Banerjee and Naumann also gave permission-dependent type system for access control mechanism but cannot capture 'non-monotonic' security policies, which are important for Android applications. Chen provided a sound type system that can capture such policies, and also proved the decidability of its type inference problem. This result was published in CSF'18.

Comments

Chen's PhD thesis is a solid piece of work in software security. He clearly made original contributions to the area, some of which were already published at top venues including CSF, CCS, and FSE.

Generally, the thesis requires some improvements in its presentation including making it more readable, and making the formatting more consistent. In addition, the authors also need to improve the motivation and conclusion parts of the thesis, to indicate that the PhD thesis is a coherent collection of work, and that there is further work to be done (which I believe to be the case). Let me make some specific comments to improve the thesis below.

Firstly, in its present state Chapter 6 (type-based verification of information flow in Android apps) does not seem to fit with the rest of the thesis, although the result on its own is a solid piece of work. Primarily this is because this is a *very specific* problem in designing correct systems, while the other chapters concern more general software security problems (testing sequential and concurrent programs). In fact, any reader would be surprised to see the substantial jump from informal presentation from previous chapters to the very formal and detailed presentation in Chapter 6, which came without warning. I believe there are two ways to go about fixing this. Firstly, one could remove Chapter 6. Instead of making the thesis weaker, I believe on the other hand that this increases the strength of the thesis, since the thesis becomes more coherent. Another way to address this problem is to give sufficient general background on verification and type checking (which the thesis is currently lacking), while mentioning that the thesis attacks this specific problem. I personally prefer the former approach.

Secondly, owing to the breadth of the currently submitted thesis, the author should provide more background to guide the reader for a smoother read. I mentioned providing more background on type-based verification in general, but also it would be useful to provide slightly more background on greybox fuzzing, directed greybox fuzzing, and the improvement provided by Hawkeye on specific small examples.

Thirdly, the presentation/formatting between various chapters should be made more consistent. For example, how come you have thread of validity in Chapter 3, but none in other chapters. Why do you have box formatting for 'Answers to Question …' in Chapter 4, but not in Chapter 3? Finally, it seems strange to me that you have experiments in Chapter 3 and Chapter 4, but no experimental results at all in Chapter 6.

Fourthly, I feel Chapter 4 (greybox for multithreaded programs) should contain more in-depth comparisons with the numerous related work on schedule-based and data-race detection tools (including Chess, Goldilocks, Eraser), which I currently feel is still lacking. Doublade aims to explore a large set of execution paths of multithreaded programs by exercising different interleavings of the threads. However, comparisons were only made with greybox fuzzers (esp. AFL), which are not specialized for target coverage with respect to thread interleavings and are not effective at "finding more concurrency faults". In particular, I strongly recommend more in-depth comparisons with the papers:

- "Race directed random testing of concurrent programs (Koushik Sen, PLDI'08)" and
- "Coverage guided systematic concurrency testing (Chao Wang et. al., ICSE'11)".

How does your technique compare with them experimentally?

Finally, the thesis still has many small typos that should be fixed before the submission of the final version. Some of these would have been caught by spell checkers. The thesis would also benefit from proof-reading by native speakers of English.

Minor and detailed comments:

1. Figure 1.1: the chapter numbers are off by +2 here.
2. Abstract and elsewhere: ‘Isomerism nature’ --perhaps→ 'multifaceted nature’?
3. Page 23, last paragraph (‘For P1’): provide more details how you get these numbers.
4. Page 27: ‘oracle seed’ (describe what this means)
5. Page 38: ‘availale’
6. Page 44: ‘an newly’
7. Section 3.6: why are there a lot of ‘.’ before the start of paragraphs?
8. Page 52: ‘conjunction’ -> ‘intersection’
9. Page 52: ‘multithreading segment’ - what does that mean? Define.
10. Page 52: ‘fun_2’ -> ‘func_2’
11. Page 53: what is N_c?
12. Listing 4.1: ‘V_c’ should be ‘V_{cb}
13. Section 4.7.2, second paragraph: ‘Dynamic detectors’ -> ‘Dynamic race detectors’. Be more specific here in the related work section!
14. Referenes: be careful with capitalization here, e.g., in [44], ‘c’ should be ‘C’