CIS 700 (Fall 2017) Project Milestone 1

Robert Zajac (rzajac), Graham Mosley (gmosley)

1. What is your project trying to accomplish?

Security flaws in programs are a huge source of vulnerabilities in larger software systems. In this project we analyze the extent to which we can detect security flaws in C/C++ programs with tools designed for this purpose as opposed to general static analysis tools. In particular we will compare and contrast the security tool Flawfinder for C/C++ against more general static analyses like Clang's static analyzer and Facebook's Infer by running these tools on real programs and analyzing what kind of vulnerabilities they are able to find. Our project ultimately aims to answer whether there are particular strategies or designs for the implementation of static analyses that predispose the tools to finding security bugs, or whether a more general class of tools is sufficient for finding these bugs. We believe this question to be of interest to the security-conscious C/C++ programmer so she can use the best tool for finding bugs in code. This is also meaningful for developers of static analyses so they are better aware of strategies that analyses may use for finding security bugs.

2. What have you done so far in the project?

As a starting point, we gathered sample C/C++ programs to test with different static analysis tools. We found https://github.com/deadbits/InsecureProgramming which is a collection of small C programs used to teach popular exploitation techniques. For more complicated C/C++ programs we decided to use previous challenges from the DARPA Grand Challenge https://github.com/trailofbits/cb-multios/tree/master/challenges.

We also found an extremely useful set of benchmarks for almost every type of C/C++ code security defect https://github.com/regehr/itc-benchmarks created by Toyota

InfoTechnology Center USA in 2014. They also published a paper (http://ieeexplore.ieee.org/document/7392027/?arnumber=7392027) using the benchmarks with three different proprietary static analysis tools. We feel that our research is still providing something novel because that static analysis field has grown significantly in the last few years and has allowed for many open source projects to mature. In particular we feel that our work will give a more in-depth understanding of which features in static analyses are critical for finding security flaws.

Once we gathered enough programs, we investigated multiple static analysis tools. We spent the most amount of time with FlawFinder, Facebook Infer, Cppcheck and the Clang Static Analyzer. We briefly ran these tools on the C programs we found to make sure that they could find results and that there wouldn't be major obstacles to conducting more in-depth runs of the tools in the future. Indeed the tools ran without issue (for the most part) and already started to report interesting results about the buggy C programs we've found.

Our progress is in a Git repository (https://github.com/gmosley/cis700sa-project).

3. What challenges have you encountered?

Originally we had planned to using static analyses on Java and Python programs to look for security vulnerabilities and compare how prominent vulnerabilities were in those languages to C/C++. After searching for more Java and Python static analysis tools as well as buggy programs in these languages, we were unable to find many results. We determined that the abstractions provided by these languages ruled out classes of security flaws that are present in C/C++. In particular the classic buffer overrun bugs prevalent in flawed C/C++ programs are necessarily not present in Java/Python programs. Our options for finding vulnerabilities in Java/Python code were reduced to looking for injections like SQL injection or bugs in the runtime environments (JVM). While these bugs are still important to rule out, we felt that static

analysis tools that find them would not be distinct enough from each other to merit a meaningful investigation. We determined that these bugs would not be as interesting to search for and so we changed the scope of our project to comparing tools for finding security bugs in C/C++ code.

We also found that the different C/C++ codebases we are investigating usually have different build systems that are not immediately compatible with some of the tools we use. That is, there is a non-trivial amount of manual intervention done by us to compile the code in accordance with what is expected by these tools.

4. What do you plan to do by the next milestone?

1. Run all the analyses we are experimenting with fully on several programs with security flaws and compile the results into a format that we can easily analyze for similarities and differences. In particular we should categorize C/C++ programs based on what kind of vulnerability they hold, run the tools on each program, and generate interesting data about the execution of these tools and their findings.
2. Investigate the source code of the tools we are using and start to get a sense of how they work. In particular we intend to isolate the different features of the implementations of these tools and categorize them based on these features. Then, we intend to compare our classification of these features with the results that each tool gave on the buggy programs. This should point us in the direction of which analysis features are critical for finding bugs.
3. Investigate the potential to automate building different codebases in the way that any particular tool expects even in the presence of different build systems. Our goal would be for this experiment to be easily reproducible so we want to eliminate as much administrative overhead in running the tools as possible.

5. Do you have any additional comments?

We had not heard feedback on the initial project proposal before compiling this milestone submission so we are not completely sure if we are on the right track in terms of expectations for the project. We hope this work is meaningful and any feedback is much appreciated!