Analysing Software Security Vulnerabilities

Jaco van de Pol, Freark van der Berg

Rules of the game:

1. Please work in groups of 2 persons, and enroll in a Blackboard group.
2. The work should be submitted on Blackboard under "Assignments".
3. Part A is an exercise (optional). Part B is the obligatory core. (C) is the "bonus parts".
4. Clearly indicate who participated in which part of the assignment.
5. The submission consists of 2 files: a "zip" (all code) and a "pdf" (report).
6. The "readme"-file contains precise instructions *including the format of the zip*. You should*must* follow this format for our automated test suite.
7. The "pdf" should contain:
   - Your names and s-numbers, and roles in the various parts
   - A short description of your approach, including how you used the tools (max 1-2 A4 per part)
   - A short description of the results and a description of the found bugs (max 1-2 A4 per part)
8. The deadline for this first assignment in Blackboard is: **December 6, 23:59 CET**.
9. Submissions that arrive either by E-mail, or arrive after the final deadline, or in wrong zip+pdf formats will be ignored by us.

A: Testing valgrind and cppcheck (exercise: optional)

The goal is to check what kind of errors or warnings are actually found by simple-to-use tools, like valgrind and cppcheck, or even by the compiler itself. You can proceed as follows:

1. Install valgrind, cppcheck, and check the version of your C-compiler.

2. Insert the code snippets from slide-set “Buffer Overflows” pages 3, 10, 24, 26, 30, 40, 46 in complete C-programs. This involves putting them in a main-function, #including the required libraries (and fixing some typos).
3. Check and compare the errors and warnings generated by the compiler, valgrind, and cppcheck.
4. Investigate the effect of increasing their “alert” level (warnings, inconclusive results, etc.).

B: Writing secure code (core part: obligatory)

The goal of this part is to write secure code involving user input and pointer operations. You will implement a simple binary search tree (SINT) on data-items (consisting of age-name pairs). See the zip file, including a readme and test program, for details and a program skeleton. We provide a (not too secure) main program taking commands for the SINT on the prompt (see the zip file). See the README for the exact operations that you need to implement.

You will inspect the code manually, and subject it to analysis by the compiler, valgrind, and cppcheck, and improve any security issues. Report on the whole process, not just the end product. Please keep your code for later. You will inspect it later by stronger analysis tools, and you will reprogram this in a modern (and more secure!) programming language.

C: Apply to an open source code base (bonus)

You can apply security analysis to existing code bases, increasing your impact. One strategy could be to focus on well-known security-sensitive projects, but the chance to find new bugs is low. Another strategy could be to focus on new projects, where the chance of success is higher. Note that the security of a system is as bad as the weakest link. Below, we focus on relatively new code on (prototype) verification tools released in a research environment, to ensure quick feedback, but you can also choose your own target.
You can proceed as follows:

1. Select some (C/C++) code base from github (`https://github.com/`).
2. Analyse its code base for memory/security-related bugs
3. Investigate if the bug(s) can be exploited maliciously
4. If applicable, submit a pull request with a fix