Name: Tori Overholtzer

Date: 12/6

CYB220 Homework 5– Static analysis tools

Objective: Practice using static analysis tools to analyze programs.

Due: Friday, Dec 6, 2024 11:59pm on CANVAS.

Turn in: This report

Points: 60 points

Requirements:

- Take a look at the 5 vulnerable programs (3 C++, 2 C code), and understand the injected vulnerabilities.
  - Programs posted on CANVAS for this homework, *HW5-VulCode.zip*.
- Run 3 different static analysis tools to analyze the posted five vulnerable programs.
- While running the tools, finish the following report.

(15 pts) Step 1: Run 3 different static analysis tools to analyze the five vulnerable programs (posted on CANVAS for this homework, HW5-VulCode.zip).

The 3 static analysis tools are:

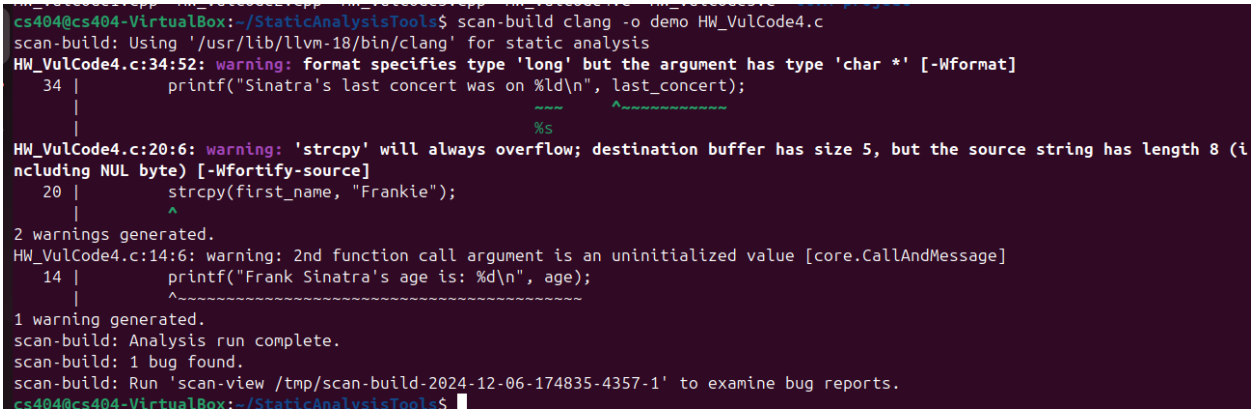
Clang/clang++ static analyzer (https://clang-analyzer.llvm.org)

Gcc/g++ static analyzer (https://gcc.gnu.org/onlinedocs//gcc/Static-Analyzer-Options.html)

Cppcheck (https://cppcheck.sourceforge.io)

STATIC ANALYSIS TOOLS #1

(15 pts) Run static analysis tool #1 Clang/clang++ static analyzer to analyze the vulnerable programs HW_VulCode4.c The result from the tool is



(A screenshot of analyzing one program is good enough; I don't need five screenshots):

Question: What is the tool good for? Not good for? Does it give you false positive results?

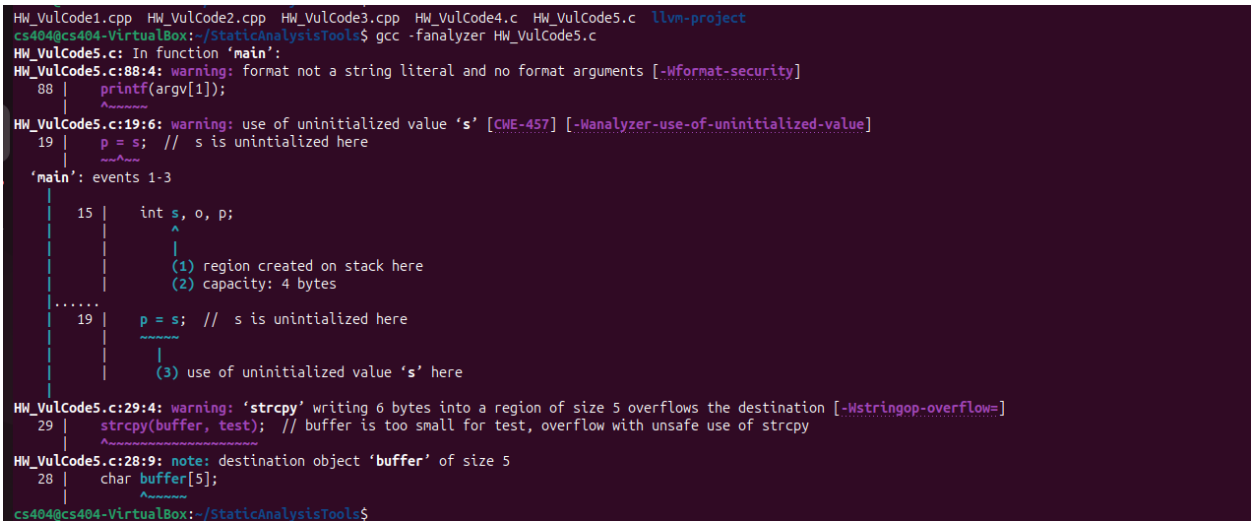
The tool is good for early detection of issues before program execution which can improve code quality, application reliability and overall security. It is accurate and works well with both C and C++. It can detect memory leaks, null pointer errors, buffer overflows, out of bounds access, and unexpected behavior. The clang static analyzer allows you to create custom checks, and its output provides detailed messages to explain what went wrong and where.

The tool is not good for ease of use- it requires setup and configuration especially on large scale projects. The analysis will be slower when analyzing large scale projects over a small-scale project. Not all logic-based issues can be detected without additional rules.

It can give false positives.

STATIC ANALYSIS TOOLS #2

(15 pts) Run static analysis tool #2 Gcc/g++ static analyzer to analyze the vulnerable program HW_VulCode5.c. The result from the tool is (take a screenshot):



https://gcc.gnu.org/onlinedocs/gcc/Static-Analyzer-Options.html

Question: What is the tool good for? Not good for? Does it give you false positive results?

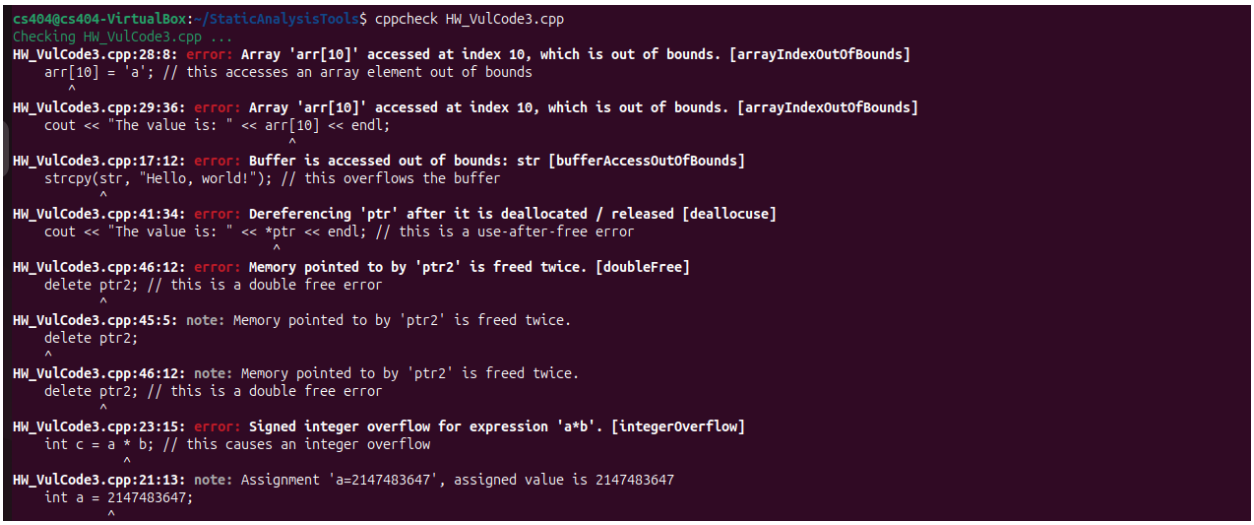
The tool is good for ease of use and setup to analyze C/C++ programs. It is apart of the GCC/G++ toolchain and developers familiar with this toolchain will find there is little to no learning curve to use the tool. It can be invoked with a flag. It has customizable options via different flags and can be ran on modules or individual files as scope requires. This static analysis tool allows issues to be caught before execution. It can identify common bugs like using uninitialized variables, buffer overflows, format string vulnerabilities, and use of unsafe functions. This ease of use promotes good programming practices and better security.

The tool is not good for detecting issues such as integer overflows, off by one errors (may show up as buffer overflow), type mismatch, use after free, and double free errors. It may also miss complex bugs. It has a limited scope of usability, working only with C/C++ programs. It may also give false positives or generate warnings-flagging issues that aren't bugs which leads to unnecessary time spent in further analysis. It lacks in depth/sophisticated analysis compared to dedicated static analysis tools (Clang Static Analyzer) and lacks advanced features such as code complexity metrics or integration with databases. It requires code to be buildable and have access to libraries because it analyzes code after successful compilation. Without access to libraries or if only part of the code is available (when using third party libraries) the analyzer might not provide enough detailed information to be useful. It has limited customization options especially around configuring scope/depth of analysis. Reporting might be excessively verbose and less user friendly to read as opposed to dedicated tools. It might not scale well for analyzing large code bases as the analyzer may struggle with performance or produce excessive output making it difficult to quickly find critical or high priority issues.

It can give you false positives.

STATIC ANALYSIS TOOLS #3

(15 pts) Run static analysis tool #3 Cppcheck to analyze the vulnerable program HW_VulCode3.cpp. The result from the tool is:



Question: What is the tool good for? Not good for? Does it give you false positive results?

The tool is good for identifying bugs, security issues and code standard violations by detecting issues around memory leaks, security vulnerabilities, null pointer dereferences, logic errors, uninitialized variable usage, and undefined behavior early in the development process before code execution. It can identify buffer overflows, division by zero errors, accessing invalid memory locations, improper handling of user input or unsafe function calls, catches null pointer access and will flag unreachable code or logical inconsistencies in control flow. It can use predefined checks or customized user check conditions and can be integrated into automated development lines or with integrated development environments (IDE). It is lightweight and a powerful tool. It is actively maintained and currently gets regular updates.

The tool is not good for ease of configuration. Customizing checks can require in depth configuration. It also doesn't scale with large code bases well. Large projects may experience slower analysis times compared to smaller scale projects with less complexity. This tool may also struggle to identify issues with highly complex code bases or code bases that use obscure language features. Because this is a static analysis tool it won't catch any errors that relate to runtime environments or dynamic testing. It may also generate false positives or generate warnings that turn out to not actually be issues.

It can give false positive results.

Summary

(15 pts) Step 2: Final summary of using the 3 static analysis tools. Compare these 3 static analysis tools. For columns 2, 3 and 4, if the tool can identify the bug, mark "YES". Otherwise, mark "NO".

Vulnerability	Clang/clang++ static analyzer	Gcc/g++ static analyzer	Cppcheck
Uninitialized variable	YES : HW_VulCode5.c	YES: HW_VulCode5.c	YES: HW_VulCode3.cpp

Buffer overflow	YES : HW_VulCode3.cpp	YES: HW_VulCode5.c	YES: HW_VulCode3.cpp
Integer overflow	NO	NO	YES: HW_VulCode3.cpp
Off-by-one	YES: HW_VulCode3.cpp	NO	YES: HW_VulCode3.cpp
Type mismatch	YES: HW_VulCode4.c	NO	NO
Use-after-free	NO	NO	YES: HW_VulCode3.cpp
Double free	NO	NO	YES: HW_VulCode3.cpp
Use of unsafe string functions	YES: HW_VulCode4.c	YES: HW_VulCode5.c	YES: HW_VulCode3.cpp
Format string vulnerability	YES : HW_VulCode3.cpp	YES: HW_VulCode5.c	NO
Summary of the tool	The Clang Static Analyzer is a tool within the LLVM ecosystem that detects potential bugs and vulnerabilities such as format string vulnerabilities, use of unsafe string functions, type mismatch, off by one errors, buffer overflows and uninitialized variables in C, C++, and Objective-C code through static analysis without executing the program.	The Gcc/G++ static analyzer is a tool within the gcc/g++ compiler toolchain that detects potential bugs and vulnerabilities such as format string vulnerabilities, use of unsafe functions, buffer overflows and use of uninitialized variables in C and C++ code through static analysis after compilation.	Cppcheck is a static analysis tool for C and C++ designed to identify bugs, security vulnerabilities, and coding inefficiencies in regards to use of unsafe functions, double free and use after free pointer violations, off by one errors, buffer overflows, integer overflows, and use of uninitialized values while providing highly customizable and versatile analysis options.